ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC SƯ PHẠM
KHOA TIN HỌC

Phan Đoàn Ngọc Phương

GIÁO TRÌNH
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Đà Nẵng - 2007

Chương 1 Tổng quan về cấu trúc dữ liệu và giải thuật

I. Khái niệm :

Xem xét các cấu trúc dữ liệu kinh điển và các cách xử lý tương ứng.

I.1. Cấu trúc dữ liệu (CTDL):

- Cấu trúc dữ liệu là những dữ liệu phức hợp gồm nhiều thành phần. Ví dụ mảng, bản ghi, tập hợp ...
- Cấu trúc dữ liệu là 1 đối tượng chỉ có một tên gọi và tồn tại một cơ chế để truy cập đến từng thành phần của đối tượng đó.
- Những điểm cần quan tâm khi xem xét một cấu trúc dữ liệu:
  - mô hình quan niệm
  - cấu trúc lưu trữ : cách thức bố trí các phần tử của cấu trúc dữ liệu bên trong bộ nhớ
  - Các phép toán cơ bản trên cấu trúc :
    + Cách thành lập cấu trúc
    + Bổ sung và loại bỏ phần tử
    + Duyệt cấu trúc (mỗi phần tử đến một lần)
    + Tìm kiếm (tìm phần tử thỏa mãn điều kiện nào đó)
    + Sắp xếp
  - Các ưu, khuyết điểm của cấu trúc đó.
- Hiệu suất của giải thuật (nếu ta xem xét giải thuật)

I.2. Giải thuật (GT) :

- Định nghĩa:

Giải thuật là một khái niệm quan trọng của toán học. Giải thuật là một dãy xác định , hữu hạn các thao tác mà sau khi thực hiện chúng một cách tuần tự ta sẽ được kết quả mong muốn. "Hữu hạn" được hiểu là cả về mặt thời gian thực hiện lẫn công cụ thực hiện. **Ví dụ:** vào phòng máy

B1: mở khoá B2: Bật đèn
B3: Bật cầu dao B4: Bật công tắc CPU
B5: Bật công tắc màn hình

Nói cách khác GT thể hiện một giải pháp cụ thể , thực hiện từng bước một, để đưa tới lời giải cho một bài toán nào đó.

Khi giải một bài toán trên máy tính điện tử (MTĐT) ta quan tâm đến thiết kế giải thuật. Nhưng cần nhớ rằng: giải thuật là đặc trưng cho cách xử lý, mà cách xử lý thì thường liên quan đến đối tượng xử lý, tức là *"dữ liệu"*. Cung cách thể hiện dữ liệu mà theo đó chúng được lưu trữ và được xử lý trong MTĐT được gọi là *cấu trúc dữ liệu* (CTDL). Như vậy giữa CTDL và giải thuật luôn có quan hệ: thay đổi CTDL sẽ dẫn đến thay đổi giải thuật.

- Các đặc trưng của giải thuật :

- Tính dừng : sau một bước hữu hạn giải thuật phải dừng.
- Tính xác định : các bước của thao tác phải rõ ràng, không gây sự nhập nhằng. Nói rõ hơn là trong cùng một điều kiện, hai bộ xử lý cùng thực hiện một bước của giải thuật phải cho kết quả như nhau.

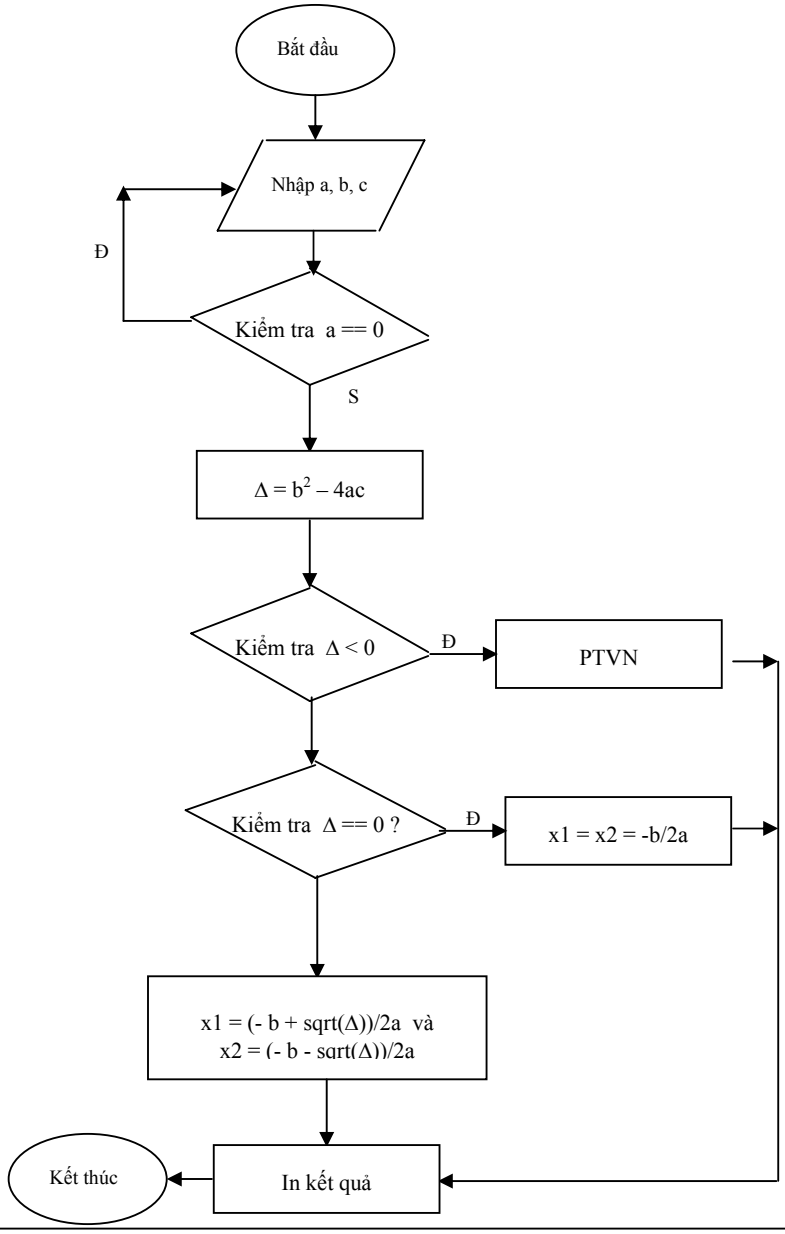
- Tính hiệu quả : giải thuật cần phải đúng đắn nghĩa là sau khi đưa dữ liệu vào giải thuật sẽ hoạt động và đưa ra kết quả như ý muốn.
- Tính phổ dụng : giải thuật có thể giải bất kỳ bài toán nào trong lớp các bài toán. Cụ thể là giải thuật có thể có các đầu vào là các bộ dự liệu khác nhau trong một miền xác định.
- Yếu tố vào ra : Một giải thuật luôn luôn có một đối tượng vào và một đối tượng ra.

I.3. Ngôn ngữ giải thuật :

Giải thuật thường được mô tả bằng một dãy các lệnh. Bộ xử lý sẽ thực hiện lệnh theo một trật tự nhất định cho đến khi gặp lệnh dừng thì kết thúc. Ngôn ngữ GT gồm 3 loại : NN liệt kê từng bước, sơ đồ khối và NN cấu trúc

Ví dụ : Giai thuật giải phương trình bậc hai $ax^2 + bx + c = 0$

- **Sơ đồ khối :**



NN liệt kê từng bước :

B1 : xác định các hệ số a, b, c

B2 : kiểm tra xem a < > 0 không ?

nếu a = 0 thì quay lại B1

B3 : Tính $\Delta = b^2 - 4ac$

B4 : nếu $\Delta < 0$ thì thông báo " PTVN" và chuyển đến B8

B5 : nếu $\Delta = 0$ thì tính x1 = x2 = -b/2a và chuyển B7

B6 : nếu $\Delta > 0$ thì tính x1 = (- b + sqrt(Δ))/2a và (- b - sqrt(Δ))/2a và chuyển B7

B7 : Thông báo các nghiệm x1, x2

B8 : kết thúc GT

- NN lập trình :

Để máy tính hiểu được GT, ta sử dụng một ngôn ngữ lập trình cụ thể để diễn đạt GT thông qua ngôn ngữ đó, cụ thể ở trong giáo trình này là ngôn ngữ C.

II. Công cụ biểu diễn giải thuật : Dựa trên ngôn ngữ lập trình C

II.1 Các lệnh vào ra : printf(…) : xuất dữ liệu

Printf(…) + scanf(..) : nhập dữ liệu

II.2 Lệnh tính toán : .

II.3 Lệnh điều khiển :

(i) if (Biểu thức logic) <Lệnh 1>;
[else <Lệnh 2> ;]

(ii) switch (biểu thức nguyên)
{
case N1: Lenh1; break ;
case N2: Lenh2; break ;
…
[default : Lenh;] break ;
}

iii) for (bt1; bt2; bt3) Lenh ;

iv) while (Biểu thức logic) <Lenh >;

v) do Lenh ;
while (biểu thức logic);

*** Chú ý** : Lệnh break ; thoát khỏi vòng lặp trong cùng.

Lệnh continue ; bỏ qua phần còn lại trong vòng lặp và thực hiện vòng lặp tiếp theo.

II.4 Khai báo: biến, hằng, kiểu dữ liệu, hàm, thủ tục ...

II.5 Hàm và chương trình : Một chương trình C bao gồm nhiều hàm. Hàm là một đơn vị độc lập, khép kín.Hàm main() là hàm bắt buộc của chương trình.

*** Chú ý :**
- Một hàm không được bao gồm các hàm khác.
- Tránh dùng biến toàn cục trong hàm
- Các biến trong mỗi hàm chỉ được sử dụng trong nội bộ hàm đó.
- Các giải thuật trong giáo trình này được trình bày dưới dạng hàm.

III. Chương trình đệ qui :

Khái niệm: Chương trình (CT) đệ qui là chương trình có chứa lời gọi đến chính bản thân nó, nghĩa là chương trình đệ qui thực hiện sau khi thực hiện bản sao của chính nó

Điều kiện lập CT đệ qui :

+ Khi bài toán có thể phát biểu thông qua chính bản thân nó, nhưng với kích thước nhỏ hơn

+ Kích thước của bài toán bằng cách này hay cách khác phải trở thành tham số (gián tiếp hoặc trực tiếp)

*** Chú ý :** - Khi gặp CT đệ qui ta phải phân biệt được các trường hợp suy biến và trường hợp đệ qui (trường hợp suy biến không gọi đệ qui nữa)

- CT đệ qui có hiệu suất kém hơn so CT không đệ qui, tuy nhiên CT đệ qui đơn giản và dễ hiểu hơn

Ví dụ 1: Tính giai thừa của một số nguyên >=0.

Phát biểu bài toán:

giaithua(n) = 1, khi n=0 // trường hợp suy biến
giaithua(n) = n * giaithua(n-1), khi n>0 // trường hợp đệ qui

CT đệ qui:

```
long  giaithua(int n)
{
  if  (n = = 0)  return  1;
  else     return (n * giaithua(n-1)) ;
}
```

Ví dụ 2: Tìm ước số chung lớn nhất của hai số nguyên dương a va b.

Phát biểu bài toán:

USCLN(a, b) = b , nếu (a % b = = 0) // trường hợp suy biến
USCLN(a, b) = USCLN(b, a % b), nếu (a % b != 0) // trường hợp đệ qui

CT đệ qui:

```
int  uscln(int a, int b)
{
  if  (a % b = = 0)  return b;
  else   return (uscln(b, a % b);
}
```

Bài tập :

1) Đọc chương 1 và 3 sách CẤU TRÚC DỮ LIỆU và GIẢI THUẬT của Đỗ xuân Lôi.

IV. Độ phức tạp của giải thuật (GT) :

IV.1 Khái niệm :

Tímh hiệu quả của GT bao gồm hai yếu tố cơ bản:

- Không gian nhớ cần thiết cho những dữ liệu vào, các kết quả tính toán trung gian và các kết quả của GT.
- Thời gian cần thiết để thực hiện GT (ta gọi là thời gian chạy CT).

Việc đánh giá hai yếu tố trên sẽ cho ta cơ sở để xác định giải thuật nào là tốt hơn. Tuy nhiên hai yếu tố trên lại hay mâu thuẩn nhau: tốt về thời gian thường lại không tốt

về không gian và ngược lại. Vì vậy trong thực tế đối với từng loại bài toán, một trong hai yếu tố sẽ được coi trọng hơn.

Thông thường thời gian thực hiện GT vẫn được chú ý hơn; vì vậy sau đây ta sẽ xét việc đánh giá thời gian thực hiện GT.

Có hai cách tiếp cận để đánh giá thời gian thực hiện của một GT. Thời gian chạy chương trình phụ thuộc vào các yếu tố chính sau:

(1) Các dữ liệu vào.

(2) chương trình dịch để chuyển chương trình nguồn thành mã máy.

(3) Tốc độ thực hiện các phép toán của máy tính được sử dụng để chạy chương trình.

Thời gian thực hiện GT chịu ảnh hưởng của nhiều yếu tố. Vì vậy ta không thể tính chính xác thời gian bằng phút, giây, ... như cách đo thời gian thông thường. Trong phương pháp lý thuyết, ta sẽ coi thời gian thực hiện GT phụ thuộc vào kích thước của dữ liệu vào hay nói cách khác nó như là hàm số của *cỡ dữ liệu* vào. Cỡ dữ liệu vào là một tham số đặc trưng cho dữ liệu vào, nó có ảnh hưởng quyết định đến thời gian thực hiện chương trình. Thông thường cỡ của dữ liệu vào là một số nguyên dương n. Ta sẽ sử dụng hàm số T(n), trong đó n là cỡ dữ liệu vào, để biểu diễn thời gian thực hiện của một GT.

Thời gian thực hiện của một GT không những phụ thuộc vào cỡ dữ liệu mà còn phụ thuộc vào dữ liệu cá biệt. Chẳng hạn, ta xét bài toán tìm kiếm một đối tượng x trên một danh sách n phần tử. Nếu xem T(n) là số phép so sánh, ta có T(n) <= n, trường hợp xấu nhất T(n) = n. Vì vậy, ta có hai cách nói là thời gian thực hiện GT trong trường hợp xấu nhất và thời gian thực hiện trung bình.

Ta có thể xác định thời gian thực hiện T(n) là *số phép toán sơ cấp* cần phải làm khi thực hiện GT. Chẳng hạn các phép toán số học +, -, *,/, và các phép toán so sánh =, <, <=, >, >=, < > là các phép toán sơ cấp. Phép toán so sánh chuỗi kí tự không thể xem là phép toán sơ cấp vì thời gian thực hiện phụ thuộc vào độ dài của chuỗi.

Tóm lại, độ phức tạp của GT là thời gian để thực hiện GT đó. GT A với kích thước đầu vào là n thì thời gian thực hiện GT được biểu diễn là T(n) và có độ phức tạp là O(f(n)) nếu tìm được 1 hằng c sao cho: T(n) <= c.f(n), với bất kỳ n >= n_0

IV. 2 Cách tính độ phức tạp :

- Q1 : một lệnh có thời gian thực hiện không phụ thuộc vào đầu vào thì lệnh đó có độ phức tạp là O(1) (hay thời gian thực hiện là hằng số)
- Q2 : Nếu lệnh b thực hiện sau lệnh a và nếu a có độ phức tạp O(f(n)) và b có độ phức tạp O(g(n)) thì độ phức tạp tổng cộng là O(max(f(n), g(n)) hay
  O(f(n), g(n)) = O(max(f(n), g(n))
- Q3 : Nếu b lồng trong a và a có độ phức tạp là O(f(n)) và b có độ phức tạp là O(g(n)) thì độ phức tạp là O(f(n)*g(n)) hay O(f(g(n))) = O(f(n)*g(n))
- **Ghi chú** : Đôi khi độ phức tạp của GT phụ thuộc vào giá trị cụ thể của dữ liệu, trong trường hợp này ta có thể xét tới độ phức tạp trong trường tốt nhấp, tồi nhất và độ phức tạp bình quân

IV.3 Một số độ phức tạp thường gặp :

Ký hi•u l•n	Tên gọi thông thường	Các phép toán
O(1)	H•ng	gán, so sánh
O(logn)	logarit	tìm nh• phân
O(n)	Tuy•n tính	tìm tuy•n tính
O(nlogn)	nlogn	QuickSort, TreeSort...
$O(n^2)$	bình ph••ng	SX ch•n, SX chèn...
$O(n^3)$	l•p ph••ng	•a th•c b•c 3
$O(2^n)$	m• (lu• ti•n)	

Ví dụ:

*) Xét đoạn chương trình sau :

```
for (i = 0; i<n; i++)
  {
        ...
  }
```

Độ phức tạp là n.0(1) = 0(n)

```
for (i = 0; i<n; i++)
  {
        for (j = 0; j<n; j++)
        ...
  }
```

Độ phức tạp là $n.0(n) = 0(n^2)$

*) Xét đoạn chương trình sau :

```
i = 1 ;
While  (i <= n)
  {
        ...
        i := i * 2 ;
  }
```

i sẽ thay đổi 1, 2, 4, 8 cho đến khi vượt n. số phép lặp khi đó là $1 + |\log_2 n|$. Do đó độ phức tạp thời gian là : $0(\log_2 n)$

*) Tìm phần tử lớn nhất trong 1 dãy hữu hạn các số nguyên

```
int   max(mangsn  *a)          //max  là phần tử lớn nhất
{
 int m ;
 m = a[0] ;
 for (i = 1; i<n; i++)
        if  (m < a[i] )  m = a[i] ;
 return m ;
}
```

Số phép so sánh cần dùng tất cả là 2(n-1) + 1 do đó độ phức tạp là 0(n)

Tóm lại: Chương trình = Cấu trúc dữ liệu + Giải thuật (Niclaus Wirth)

Chương 2 Cấu trúc **Mảng**

0. Tổng quan:

0.1 Mô hình quan niệm : Mảng là 1 dãy có thứ tự (về mặt vị trí) các phần tử với 2 đặc điểm sau:

- Số lượng phần tử cố định
- Mọi phần tử đều có cùng kiểu dữ liệu (dữ liệu cơ sở của mảng)

0.2 Cấu trúc lưu trữ :

Các phần tử được bố trí sát nhau trong bộ nhớ và theo thứ tự tăng dần của các chỉ số nên dễ dàng tìm được địa chỉ của 1 phần tử bất kỳ nếu biết chỉ số:

$$Loc(a[i]) = a_0 + (i-1) * l$$

a_0 là địa chỉ của phần tử thứ nhất ; l là độ dài 1 ô nhớ (byte)

0.3 Các đặc trưng cơ bản :

+ Cho phép truy cập ngẫu nhiên đến từng phần tử. Thời gian truy cập đến mọi phần tử đều bằng nhau.
+ Số lượng phần tử của mảng là cố định. Việc bổ sung và loại bỏ phần tử là khó khăn (mất thời gian)

0.4 Các phép toán cơ bản :

Tạo mảng, duyệt mảng, tìm kiếm, sắp xếp, trộn mảng, tách mảng …

I. Tạo mảng:

I.1 Khai báo:

- Cú pháp: <Kiểu dữ liệu> Tênmảng[số phần tử lớn nhất]

hoặc <Kiểu dữ liệu> Tênmảng[]

- Khai báo mảng số nguyên: int m[50] ; hoặc int m[] ;
- Khai báo mảng sinh viên có tối đa 100 sinh viên:

```
struct sv {
  char malop[5];
  char hoten[25];
  float diem[3];
  } danhsach[100];
```

I.2 Nhập mảng:

*) Nhập mảng số nguyên từ bàn phím:

```
void nhapmang(int *m,int *n)
{
  int i;
  printf("\n Cho biet so phan tu cua mang :");
  scanf("%d",n);
  for (i=0;i<*n;i++)
   {
     printf("\n nhap phan tu thu %d :",i);
     scanf("%d",m[i]);
   }
}
```

*) Nhập mảng số nguyên bằng cách lấy ngẫu nhiên:

```
void nhapmang_ngaunhien(int *m,int *n)
{
  int i;
  printf("\n Cho biet so phan tu cua mang :");
  scanf("%d",n);
  randomize();
  for (i=0;i<*n;i++)
      m[i]=random(100);
}
```

*) Nhập mảng số nguyên mà sau khi nhập xong thì mảng đã được sắp xếp tăng dần:

```
void nhapSapXep(int *m,int *n)
{
  int i, j, tam;
  printf("\nCho so phan tu cua mang:");
  scanf("%d",n);
  printf("Nhap cac phan tu: ");
  for (i=0;i<*n;i++)
  {
    scanf("%d",&tam);
    j=0;
    while (j<i && m[j]<tam) j++;
    if (j<i)
      memmove(&m[j+1],&m[j], (i-j)*sizeof(int));
    m[j]=tam;
  }
}
```

II. Duyệt mảng :

II.1 Khái niệm : duyệt mảng tức là "thăm" các PT của mảng, mỗi PT "thăm" 1 lần.
"Thăm" : truy cập đến PT nào đó sau đó xử lý

II.2 Phương pháp duyệt chính tắc :

Giải thuật : - bắt đầu từ PT đầu tiên
- lần lượt thăm các PT theo thứ tự tăng dần của chỉ số

Nếu m : mảng [1..n] thì sẽ thăm lần lượt m[i] , i = 1..n

```
void xemmang(int *m,int n)
 {
  int i;
  printf("\n");
  for (i=0;i<n;i++)
    printf("%4d",m[i]);
  printf("\n");
 }
```

ví dụ 1 : `int a[30]; /* a la mang so nguyen */`

Tìm Sa (tổng các phần tử âm), Sd (tổng các phần tử dương), So (số lượng các phần tử = 0) của mảng a.

GT : duyệt mảng a, khi thăm 1 PT thì tùy điều kiện mà sửa lại giá trị của Sa, Sd, So

```
void tong(int *a,int n)
 {
     int i,Sa, Sd, So;
     Sa = 0 ; Sd = 0 ; So = 0 ;
     for (i=0;i<n;i++)
       if a[i] < 0 then Sa = Sa + a[i]
       else if a[i] > 0 then Sd = Sd + 1
            Else So = So + 1 ;
     printf("Tong cac phan tu am la:%4d",Sa);
     printf("Tong cac phan tu duong la:%4d",Sd);
     printf("Tong cac phan tu bang khong la:%4d",So);
 }
```

Ví dụ 2 : a là ma trận nguyên cấp n x n ;Tính tổng các PT trên đường chéo chính của a.
GT : có 2 phương án :
PA1 : duyệt toàn bộ ma trận a rồi kiểm tra nếu thuộc đường chéo chính thì cộng dồn vào S. ĐỘ PHỨC TẠP là $O(n^2)$
PA2 : duyệt các PT trên đường chéo chính mà thôi. ĐỘ PHỨC TẠP là O(n)

II.3 Duyệt tự do :

GT : duyệt các PT theo 1 trình tự logic sao cho mọi PT đều được thăm và không có PT nào được thăm quá 1 lần.

Ví dụ : Lập ma trận xoắn cấp m x n.

GT : < cho $a_{ij} = 0$, với bất kỳ i,j >
Khởi động : i =1, j = 1 , s = 1 , a[i,j] = 1
(i, j) : tọa độ của điểm hiện tại
s là giá trị của a_{ij} và là phần tử đã gieo

```
while (s < m * n)
{       <sang phải>
        while (( j + i <= n) && (a[i][ j] = = 0)
          {
               j++ ; s ++ ; a[i][ j] = s ;
          }

        <xuống dưới> < sang trái> <lên trên>
}
```

III. Tìm kiếm tuần tự:

III.1 Bài toán :

Cho mảng số nguyên `int a[30];` và một số nguyên x. Tìm chỉ số i để a[i] = x

III.2 GT cơ bản :

- Bắt đầu từ PT đầu tiên
- tìm cách đi sang phải : nếu thỏa mãn 2 điều kiện chưa vượt quá giới hạn mảng và PT đang xét khác với x
- Tùy theo vị trí của phần tử đang xét ta có kết luận : hoặc có lời giải là chỉ số phần tử đang xét, hoặc không có lời giải, ta qui ước lời giải = 0 trong trường hợp này

GT :

i := 0 ; { i là tọa độ của phần tử hiện tại}
While ((i < n) && (a[i] != s)) i = i + 1 ; {sang phải}
Nếu i < n --> lời giải = i+1
Ngược lại lời giải = 0 {qui ước}

Cài đặt :

```
int tktt(int a[],int x)
{
 int i=0;
 while (i<n && a[i]!=x)
   i++;
 if (i<n)
   return i+1;
 else
   return 0;
}
```

ĐỘ PHỨC TẠP : - Trường hợp tốt nhất : a[0] = x --> 1 phép so sánh O(1)
- Trường hợp tồi nhất : kết quả 0 --> n phép so sánh O(n)
- Bình quân cần (n+1)/2 phép so sánh --> O(n)

III.3 Kỹ thuật dùng phần tử cầm canh :

- mục tiêu : đơn giản hóa điều kiện vòng lặp, cụ thể loại bỏ điều kiện i < n
- cách làm : mượn thêm 1 phần tử nữa là a[n] và đặt a[n] = x , khi đó a[n] được gọi là PT cầm canh
- Đánh giá : tăng đáng kể tốc độ thực hiện vòng lặp nhưng tốn kém thêm 1 ô nhớ

```
int tktt_camcanh(int a[],int x)
{
 int i=0;a[n]=x;
 while (a[i]!=x)
   i++;
 if (i<n)
   return i+1;
 else
   return 0;
}
```

IV. Tìm kiếm nhị phân (Binary Search):

IV.1 Điều kiện áp dụng :dãy $a_0, a_1, a_2, a_3, ..., a_{n-1}$ phải có thứ tự, giả sử sắp xếp tăng dần tức là $a_0 <= a_1 <= a_2 <= a_3 <= ... <= a_{n-1}$

IV.2 Giải thuật tìm s trên mảng a[l..r]

GT mức 0 :

1) Nếu đoạn a[l..r] không có PT nào thì lời giải = 0
2) Ngược lại
   2.1 Đặt g = (l + r) / 2
   2.2 Nếu s = a[g] thì lời giải = g

Ngược lại nếu s < a[g] thì s không thể thuộc đoạn a[g..r] bài toán qui về tìm s trên đoạn a[l..g-1]

Ngược lại s không thuộc a[l..g] bài toán qui về tìm s trên đoạn a[g+1..r]

GT mức 1 : (không đệ qui)

1) l =0 ; r = n ; g = (l + r) / 2 ;
2) while ((l <= r) && (a[g] < > s))
   {
     nếu s < a[g] thì r = g - 1
     ngược lại l = g + 1 ;
     g = (l + r) / 2 ;
   }
3) Nếu l <= r thì lời giải = g
   ngược lại lời giải = 0 ;

```
int tknp(int a[],int s)
{ int i, start=0,end,found=0;
  end=n;
  do {
    i=(end+start)/2;
    if (a[i]==s)
      found=1;
    else if (a[i]<s )
       start=i+1;
    else if (a[i]>s)
       end=i-1;
    } while (!found && end>=start);
  if (found)
     return i+1;
  else
     return 0;
}
```

IV.3 ĐỘ PHỨC TẠP của GT :

- Cas tốt nhất xảy ra khi s = a[n/2] --> ĐỘ PHỨC TẠP là O(1) { 1 phép so sánh }
- Trường hợp xấu nhất : xảy ra khi s không thuộc a[start..end], lúc này cần 1 + $\log_2 n$ phép so sánh --> ĐỘ PHỨC TẠP là $O(\log_2 n)$
- ĐỘ PHỨC TẠP bình quân là $O(\log_2 n)$

V. Tìm kiếm bằng phương pháp nội suy :

V.1 Điều kiện áp dụng :

- Dãy a[l..r] có thứ tự
- Các giá trị khóa (a_i) phân bố đều trên đoạn a[l..r]

V.2 Giải thuật : tương đương tìm kiếm nhị phân nhưng giá trị g được xác định theo công thức :

$$g = (s - a[r])\ (r - l + 1) / (a[r] - a[l] + 1)$$

ĐỘ PHỨC TẠP được chứng minh là : $O(\log_2(\log_2 n))$

VI. Sắp xếp bằng phương pháp chọn (Selection Sort) :

VI.1 Bài toán : Cho mảng a[n]. Cần hoán vị các PT của mảng a để chúng trở thành có thứ tự tức là $a_0 <= a_1 <= ... <= a_{n-1}$
Giả sử ta có hàm sau để hoán vị 2 giá trị a và b:

```
void hv(int *a, int *b)
{
  int t;
  t=*a;
  *a=*b;
  *b=t;
}
```

VI.2 Giải thuật sắp xếp chọn : (SX tăng dần)
Mức 0 : - tìm PT nhỏ nhất của dãy
- hoán vị PT nhỏ nhất với PT đầu tiên
--> Vậy PT đầu tiên đã đúng vị trí. Bài toán qui về sắp xếp dãy a[1..n-1] với chiến thuật như trên
Mức 1 : Sắp xếp dãy a[0..n-1] qua n - 1 bước với i = 0, n -2
ở bước thứ i :
*) Tình trạng (tình trạng dữ liệu trước khi thực hiên bước thứ i) Dãy a[0..n-1] gồm 2 phần:
+ dãy con trái a[0..i -1] gồm các phần tử đầu cố định vị trí gọi là dãy đích.
+ dãy con phải a[i..n-1] gồm các phần tử cuối không cố định vị trí và cần sắp xếp gọi là dãy nguồn.
*) Thao tác : - Tìm j để a_j là phần tử nhỏ nhất của dãy nguồn a[i..n-1]
- Hoán vị a_i và a_j

Minh họa:
Ta cần sắp xếp dãy số sau: 44 55 12 42 94 18 6 67

i = 0 44 55 12 42 94 18 6 67

i = 1 6 : 55 12 42 94 18 44 67

i = 2 6 12 : 55 42 94 18 44 67

i = 3 6 12 18 : 42 94 55 44 67

i = 4 6 12 18 42 : 94 55 44 67

i = 5 6 12 18 42 44 : 55 94 67

i = 6 6 12 18 42 44 55 : 94 67

6 12 18 42 44 55 67 94

VI.3 Cài đặt :

```
void selectionsort(int *a, int n)
{
 int i,j,k;
 for (i=0;i<n-1;i++)
   {
     j=i;
     for (k=j+1;k<n;k++)
      if (a[j]>a[k]) j=k;
     hv(&a[i],&a[j]);
   }
}
```

ĐỘ PHỨC TẠP:
+ Số phép so sánh = (n -1) + (n -2) + ... + 2 + 1 = n(n -1) / 2
+ Số phép hoán vị : n -1
---> ĐỘ PHỨC TẠP là $O(n^2)$
+ Nhận xét : Số lượng phép hoán vị tối thiểu so với mọi GT sắp xếp. Suy ra trong những trường hợp kích thước dữ liệu (kích thước 1 phần tử a[i]) là rất lớn so với kích thước khóa (tức là phần dữ liệu được so sánh) thì sắp xếp chọn là phương pháp tốt.

VII. Sắp xếp bằng phương pháp chèn (Insertion Sort) :

VII.1 Giải thuật :
* Mức 0 :
- Lấy PT đầu tiên của dãy nguồn và chèn PT đó vào dãy đích sao cho dãy đích có thứ tự
- Lặp lại bước trên cho đến khi dãy nguồn "cạn"
* Mức 1 : SX dãy a[0..n-1] qua n -1 bước, với i = 0, n -2
Ở bước thứ i :
+ Tình trạng :
- dãy đích là a[0..i] gồm những phần tử đã có thứ tự nội bộ
- dãy nguồn a[i +1..n] gồm các phần tử chưa được xem xét
+ Thao tác :
- Lấy phần tử đầu tiên khỏi dãy nguồn $x = a_{i+1}$. Vị trí thứ i +1 được xem là lổ hổng và được phép vào dãy đích
- Chèn x vào vị trí thích hợp của dãy đích a[0..i +1] để dãy đích có thứ tự

Minh họa: Ta cần sắp xếp dãy số sau: 44 55 12 42 94 18 6 67

i = 0 44 : 55 12 42 94 18 06 67

i = 1 44 55 : 12 42 94 18 06 67

i = 2 12 44 55 : 42 94 18 06 67

i = 3 12 42 44 55 : 94 18 06 67

i = 4 12 42 44 55 94 : 18 06 67

i = 5 12 18 42 44 55 94 : 06 67

i = 6 06 12 18 42 44 55 94 : 67

06 12 18 42 44 55 67 94

VII.2 Cài đặt :

```
void insertionsort(int *a, int n)
{
  int i,j,x;
  for (i=0;i<n-1;i++)
   {
     x=a[i+1];
     j=i+1;
     while ((j>0) && (x<a[j-1]))
       {
         a[j]=a[j-1];
         j--;
       }
     a[j]=x;
   }
}
```

VII.3 Nhận xét :

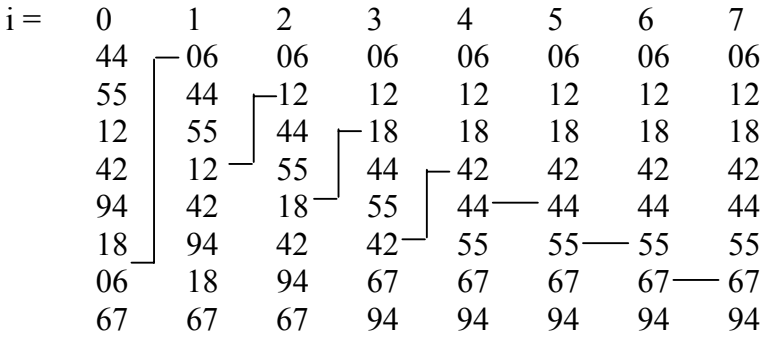
- Độ phức tạp của GT :
  + Trường hợp tốt nhất :xảy ra khi dãy ban đầu đã có thứ tự
    Cần n -1 phép so sánh, n -1 phép hoán vị
  + Trường hợp xấu nhất : xảy ra khi dãy ban đầu có thứ tự ngược lại
    Số phép so sánh : $2 + 3 + 4 + ... + n = n(n+1) / 2$
    Số phép hoán vị : $n(n-1) / 4$
  + Trường hợp bình quân : $O(n^2)$

Bài tập : Minh họa hoạt động của GT sắp xếp chọn và chèn với dãy khóa cho trước

VIII. Sắp xếp bằng phương pháp nổi bọt (Buble Sort) :

VIII.1 Giải thuật : Sắp xếp dãy a[0..n-1] tiến hành n -1 bước với i = 0..n-2
Ở bước thứ i :
* Tình trạng : - Dãy phải là a[i .. n-1] chưa có thứ tự
- Dãy traí là a[0.. i-1] gồm những phần tử đã ổn định vị trí
* Thao tác : Khi so sánh a[j-1] và a[j] (j = n-1.. i +1) hoán vị chúng nếu chúng bị ngược thứ tự, suy ra a[i] là phần tử bé nhất của a[i.. n -1]

Minh họa: Ta cần sắp xếp dãy số sau: 44 55 12 42 94 18 06 67



VIII.2 Cài đặt :

```
void bublesort()
 {
  int i,j,k;
  for (i=0;i<n;i++)
    for (j=n-1;j>i;j--)
      {
       if (a[j-1]>a[j])
      {
        k=a[j-1];
        a[j-1]=a[j];
        a[j]=k;
      }
       }
 }
```

VIII.3 Nhận xét :

- Độ phức tạp: $O(n^2)$
- Cần chính xác (n -1) + (n - 2) + ... + 1 = n(n -1) / 2
- Cần tối thiểu 0 phép hoán vị, tối đa n(n -1) / 2 phép hoán vị
- Trung bình : chậm hơn sắp xếp chọn và chèn

IX. Sắp xếp bằng phương pháp phân hoạch (QuickSort) :

IX.1 Bài toán phân hoạch : Cho dãy a[l..r] , cần phân hoạch dãy đó thành 2 dãy con sao cho 1 PT bất kỳ của dãy con bên trái <= 1 PT bất kỳ của dãy con bên phải, nghĩa là :

- Chọn x thuộc a[l .. r]
- Tìm cách xác định : dãy trái gồm các PT <= x, dãy phải gồm các PT >= x

1. Đặt i = l coi i là con trỏ duyệt dãy từ trái sang
2. Đặt j = r coi j là con trỏ duyệt dãy từ phải sang
3. Lặp lại khi i <= j
   - 3.1 tăng i cho đến khi có a[i] >= x
   - 3.2 giảm j cho đến khi có a[j] <= x { x : PT cầm canh đối với i & j }
   - 3.3 nếu i <= j thì + hoán vị a[i], a[j] + tăng i + giảm j

IX.2 Giải thuật SX bằng phân hoạch :

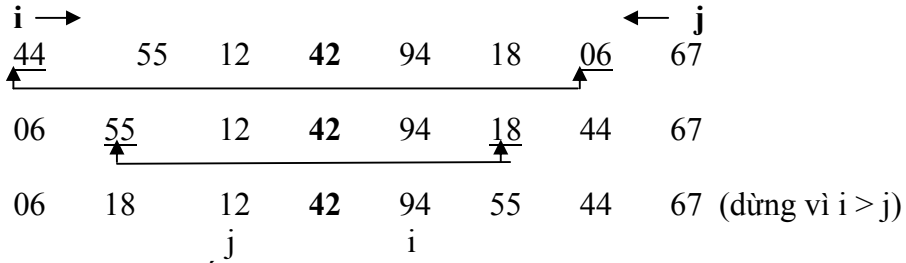
Để SX dãy a[l..r] ta phân hoạch thành 2 dãy con với tính chất như ở trên, sau đó tiến hành SX riêng dãy con trái và dãy con phải cũng bằng cách thức trên. Quá trình SX chấm dứt khi dãy con cần SX còn ít hơn 2 PT.

Minh họa: Ta cần sắp xếp dãy số sau: 44 55 12 42 94 18 06 67

Ta qui ước chọn khóa chốt là số ở giữa của dãy, cụ thể là

x=a[(i+j)/2]= 42 v•i i=l và j=r

Sau đây là diễn biến của một bước đầu:



Như vậy dãy số đã được chia làm hai phân đoạn và 42 đã ở đúng vị trí của nó. Quá trình sẽ được lặp lại tương tự với từng phân đoạn cho đến khi dãy số được sắp xếp hoàn toàn. Mỗi lần dãy được phân thành hai dãy con thì việc xử lý tiếp theo sẽ được thực hiện hoặc bằng giải thuật lặp (dùng stack) hoặc bằng giải thuật đệ qui.

IX.3 Cài đặt giải thuật đệ qui:

```
void sort(int l, int r)
 {
   int i,j,x,w;
   i=l;
   j=r;
   x=a[(i+j)/2];
   do {
    while (a[i]<x) /* 1 */
      i++;
    while (a[j]>x)
      j--;
   if (i<=j)
    {
      w=a[i];
      a[i]=a[j];
      a[j]=w;
      i++;
      j--;
    }
  } while (i<=j);
  if (l<j)
    sort(l,j);
  if (i<r)
     sort(i,r);
}
```

```
void quicksort()
{
  sort(0,n-1);
}
```

IX.4 Nhận xét GT QuickSort :

1. Tại điểm 1) không thể thay Đkiện a[i] < x và a[j] > x bằng đk a[i] <= x và a[j] >= x vì x là PT cầm canh cho quá trình duyệt của i&j (vd mảng 5 5 5 5 5 i&j sẽ đi quá giới hạn mảng)
2. Tại đk i<=j không thể thay bằng i < j vì inc(i) & dec(j) có thể dẫn tới vòng lặp vô hạn)
3. Độ phức tạp của GT :

+ Trường hợp tốt nhất : xảy ra khi ta luôn luôn tìm được x phần tử trung vị làm chốt (PT có giá trị giữa). Vậy bài toán cấp n luôn được đưa về 2 bài toán cấp n / 2 sau khoảng n phép hoán vị. Suy ra cần $n\log_2 n$ phép so sánh --> ĐỘ PHỨC TẠP là $O(n\log_2 n)$

+ Trường hợp tồi nhất xảy ra khi ta luôn chọn phải x là PT lớn nhất hoặc nhỏ nhất. Lúc này bài toán cấp n qui về bài toán cấp n -1 sau n phép so sánh. Vậy ĐỘ PHỨC TẠP là $O(n^2)$

+ Trường hợp bình quân : ĐỘ PHỨC TẠP là $O(n\log_2 n)$

4. Cải tiến :

+ Khi n đủ nhỏ (n < 20) chuyển sang phương pháp chọn hoặc chèn
+ Khử đệ qui bằng cách dùng ngăn xếp
+ Cải tiến cách tìm phần tử chốt (lấy trung vị là 3 phần tử đầu, cuối và giữa)

X. Chèn, xoá phần tử trong mảng:

X.1 Chèn 1 phần tử vào mảng:

Bài toán: Cho mảng số nguyên a[0..n-1]. Chèn một phần tử vào sau một phần tử x cho trước trong mảng. Nếu x không có trong mảng thì sẽ không chèn. Số được chèn sẽ được lấy ngẫu nhiên.

```
void chen(int *a,int *n)
{
 int i=0, x;
 printf("\nCho biet so can chen sau:");
 scanf("%d",&x); randomize();
 while (i<*n)
    if (a[i]==x)
     {
       (*n)++;
       memmove(&a[i+1],&a[i],sizeof(int)*(*n-i));
       a[i]=random(100);
       break;
     }
    else
     i++;
 }
```

X.2 Xoá phần tử trong mảng:

Bài toán: Cho mảng số nguyên a[0..n-1]. Hãy xoá phần tử có dữ liệu là x có trong mảng với x được nhập từ bàn phím.

```
void xoa(int *m,int *n)
{
 int i=0, x;
 printf("\nCho biet so can xoa:");
 scanf("%d",&x);
 while (i<*n)
    if (a[i]==x)
     {
       (*n)--;
       memcpy(&a[i],&a[i+1],sizeof(int)*(*n-i));
       break;
     }
    else
     i++;
 }
```

XI. Trộn hai mảng đã được sắp xếp:

Bài toán: Giả sử có hai mảng m[0..n-1] và mảng a[0..b-1] đã sắp xếp tăng dần. Hãy trộn hai mảng này thành mảng c[0..n+b-1] cũng được sắp xếp tăng dần.

```
void tron2mangtang()
{
 int c[50],i1,i2,i;
 nhapmang(m,&n);
 insertionsort(m,n);
 xemmang(m,n);
 nhapmang(a,&b);
 insertionsort(a,b);
 xemmang(a,b);
 i1=i2=0;
 for (i=0;i<n+b;i++)
 {
  if (i1>=n || i2>=b)
     break;
  if (m[i1]< a[i2])
  {
   c[i]=m[i1];
   i1++;
  }
  else
  {
   c[i]=a[i2];
   i2++;
```

```
  }
 }
 if (i1<n)
  while (i1<n)
   c[i++]=m[i1++];
 if (i2<b)
  while (i2<b)
   c[i++]=a[i2++];
 printf("\nCac PT cua mang tron la:\n");
 xemmang(c,n+b);
 getch();
}
```

XII.Kiểm tra mảng tăng:

Bài toán: Cho mảng số nguyên a[0..n-1]. Viết hàm kiểm tra mảng này có tăng dần không ?

```
 void daytang(int *a,int n)
 {
  int i=1;
  while (a[i]>=a[i-1] && i<n)
     i++;
  if (i==n)
   printf("\nDay da tang");
  else
   printf("\nDay khong tang");
  getch();
 }
```

Chương 3 Danh sách liên kết (DSLK)

I. Tổng quan về danh sách liên kết :

I.1 Khái niệm : DSLK là danh sách gồm nhiều phần tử (PT) thỏa mãn điều kiện :

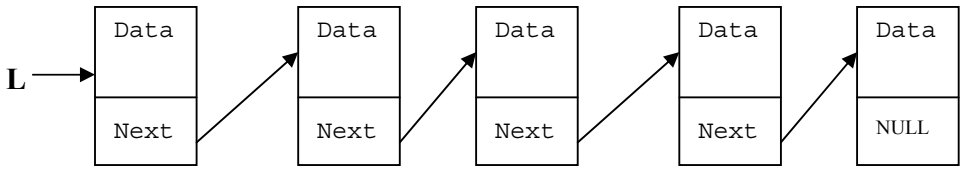
- Các PT đều thuộc cùng một kiểu dữ liệu.
- Mỗi PT ngoài phần dữ liệu bản thân nó còn chứa thông tin về vị trí của PT đứng ngay sau nó.
- Vị trí của PT đầu tiên được biết. PT đầu tiên được đại diện cho toàn bộ danh sách.

* Chú ý : Ta chấp nhận 1 danh sách đặc biệt gọi là danh sách rỗng, là danh sách không chứa PT nào. PT cuối cùng của DS phải chứa thông tin đặc biệt nhằm thể hiện rõ vấn đề này

I.2 Biểu diễn :

* Mỗi PT sẽ có :
  - Phần dữ liệu (đơn giản là 1 số nguyên - khóa)
  - Phần thông tin về vị trí tiếp theo của PT thể hiện bằng con trỏ. Giá trị NULL được sử dụng cho PT cuối

* Danh sách sẽ được đại diện bằng 1 con trỏ (trỏ đến PT đầu) Con trỏ NULL được sử dụng cho DS rỗng.




```
typedef struct node *nodep;
typedef int infotype;
struct node {
          infotype data;
          nodep next;
         };
nodep  L ;
```

I.3 Các đặc điểm cơ bản :

1. Truy cập tuần tự đến các phần tử (muốn truy cập phần tử thứ k thì phải duyệt qua k - 1 phần tử trước nó)
2. Thuận lợi cho việc thêm, bớt PT

II Các phép toán trên danh sách :

II.0 Lập DS rỗng :

```
void init(nodep *L)
{
  *L = NULL;
}
```

II.1 Kiểm tra DS rỗng :

```
int     IsEmptyList (nodep L)
{
        if  (L = = NULL) return 1 ;
        else  return 0;
}
```

II.2 Tạo danh sách:

```
void taodsFiFo(nodep *L)     // nhập ngẫu nhiên trong
[0,100)
{
 nodep p,tmp;
 char tl;
 randomize();init(L);
 printf("\n Bam fim bat ky de nhap, Bam ESC de thoi");
 do {
   p=(nodep)malloc(sizeof(struct node));
   p->data=random(100);
   p->next=NULL;
   if (*L==NULL) *L=p;
   else     tmp->next=p;
   tmp=p;
   tl=getch();
  } while (tl!=27);
}
```

II.3 Duyệt danh sách (DS):

1 Bài toán : Cho danh sách L, cần phải thăm các phần tử của DS này, mỗi phần tử thăm 1 lần

2 Giải thuật : Lần lượt thăm các PT theo đúng thứ tự trong danh sách

3. Cài đặt chương trình xem DS:

```
void xemds(nodep L)
{
 nodep p;
 p=L;
 while (p!=NULL)
 {
   printf("%4d",p->data);
   p=p->next;
 }
 printf("\n");getch();
}
```

Ví dụ : Viết chương trình tính tổng các PT của 1 danh sách L cho trước

```
int     tong(nodep L)
{       int     T ;
        T := 0;
```

```
            While  (L)
              {
                    T = T + L->info ;
                    L := L->next ;
              }
      return  T;
      }
      int     tong(nodep L)
      {
            if  (L = = nil)    return 0 ;
            else     return(L->info + tong(L->next)) ;
      }
```

II.4 Thêm 1 phần tử (PT) vào danh sách (DS) :

II.4.1 Thêm 1 phần tử vào phía sau 1 phần tử của DS :

1. Bài toán : Cho DS L và p là 1 PT của DS. Cần thêm 1 PT có khóa = e vào ngay **sau** PT p

(chú ý : PT p chính là con trỏ trỏ tới PT)

2. Giải thuật :

(i) Nếu L = NULL thì phần tử thêm vào thành phần tử duy nhất của DS
- cấp phát bộ nhớ cho 1 PT (gọi là q)
- Đưa e vào q
- Cho L trỏ tới q (để xác định là PT đầu tiên của DS)
- Cho q^.next = NULL (để xác định là PT cuối của DS)

(ii) Nếu L < > NULL
- cấp phát bộ nhớ cho 1 PT (gọi là q)
- Đưa e vào q
- Liên kết q với PT sau p
- liên kết p với q

3. Cài đặt :

```
void insert_after(infotype e,nodep p,nodep *L)
{
 nodep q;
 q=(nodep)malloc(sizeof(struct node));
 q->data=e;
 if (L== NULL)
     { L=q; q->next=NULL; }
 else
     {
       q->next = p->next;
       p->next = q;
     }
}
```

II.4.2 Thêm 1 phần tử (PT) vào phía trước 1 PT của DS :

1. Bài toán : Thêm PT có khóa e vào phía trước PT p của DS L.

2. Giải thuật : (Trường hợp L != NULL)

- nodep q;
  q=(nodep)malloc(sizeof(struct node));
  q->data=e;
- Thêm PT q vào sau p như trường hợp trên
- Hoán vị phần dữ liệu của p và q

II.4.3 Thêm 1 PT vào vị trí thứ k trong DS :

1. Bài toán : Thêm PT có khóa e vào vị trí thứ k của DS L với k là tham số của hàm.
2. Giải thuật:

```
- Tìm con trỏ tt và t lần lượt trỏ vào PT thứ k-1 và thứ k
- Nếu chèn được thì:
  + p=(nodep)malloc(sizeof(struct node));
    p->data=e;
  + Nếu (k==1) /* chèn vào vị trí đầu tiên */
           thì { p->next=t; *L=p; }
   + Ngược lại nếu(t==NULL) /* chèn vào vị trí cuốI */
           thì { p->next=NULL; tt->next=p;}
   + Ngược lại   /* chèn vào giữa */
         thì { p->next=t; tt->next=p;}
- Ngược lại thì không chèn được (k<=0 hoặc k> số PT của DS
+1)
```

3. Cài đặt:

```
void insert_mid(infotype e,int k, nodep *L)
{
 nodep p,tt,t;
 int i=1;
 t=*L;
 p=(nodep)malloc(sizeof(struct node));
 p->data=e;
 while(t!=NULL && i<k)
 {
  i++;
  tt=t;
  t=t->next;
 }
 if (i==k)
 {
 if (k==1)
 {
  p->next=t;
  *L=p;
 }
 else
  if (t==NULL)
  {
```

```
   p->next=NULL;
   tt->next=p;
  }
  else
  {
   p->next=t;
   tt->next=p;
  }
 }
 else printf("\n Khong chen duoc !");
}
```

II.5 Loại bỏ phần tử (PT) ra khỏi DS :

II.5.1 Loại bỏ PT đứng sau PT p của DS L :

1. Bài toán : Loại bỏ PT ngay sau phần tử p khỏi DS L

2. Giải thuật :

(i) Nếu L = NULL hoặc không có PT sau p thì không xoá được, hàm trả về 0

(ii) Ngược lại : - đặt q = PT sau p
- nối p với PT sau q
- hàm trả về q->data
- giải phóng q

3. Cài đặt :

```
int     ListDel ( nodep L, nodep p)
 {
        nodep q ;
        if ((L = = NULL) || (p->next = = NULL)) return 0;
        else {
                q = p->next ;
                p->next = q->next ;
                return (q->data) ;
                free(q) ;
               }
 }
```

II.5.2 Loại bỏ PT thứ k khỏi DS L :

1. Giải thuật : Các trường hợp :

+ Tìm tp và p lần lượt trỏ vào PT thứ k-1 và thứ k
+ Nếu (p= =NULL) thì không xoá được vì k<1 hoặc k>số phần tử của DS
+ Ngược lại:
  - nếu (k= =1) L = p->next / xoá PT đầu tiên */
  - ngược lại tp->next = p->next /* xoá PT ở giữa hoặc PT cuối */
+ Giải phóng vùng nhớ cho p

2. Cài đặt :

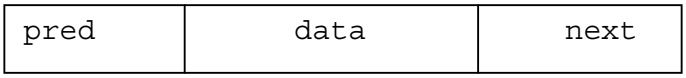
```
void del(int k,nodep *L)
{
 int i=1;
```

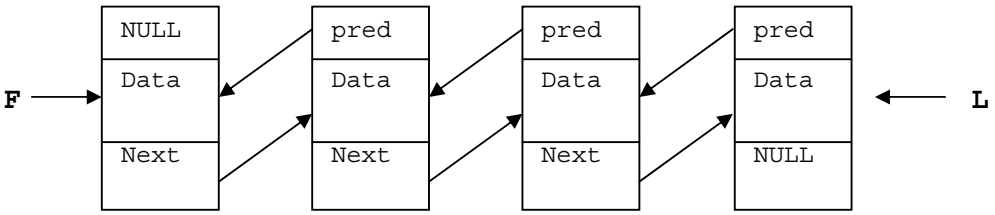
```
 nodep p,tp;
 p=*L;
 while (p!=NULL && i<k)
 {
  i++;
  tp=p;
  p=p->next;
 }
 if (p==NULL) printf("\n Khong co de xoa");
 else
 {
  if (k==1) *L=p->next;
  else  tp->next=p->next;
 }
 free(p);
}
```

III.Danh sách liên kết (DSLK) hai chiều :

III.1 Định nghĩa : DSLK hai chiều cũng giống như DSLK một chiều nhưng chỉ khác là trong mỗi bản ghi ngoài phần dữ liệu còn có 2 trường con trỏ : một con trỏ để giữ địa chỉ của bản ghi liền trước nó gọi là liên kết ngược (pred) và một con trỏ để trỏ vão bản ghi liền sau nó, gọi là liên kết thuận (next).

Mỗi phần tử của danh sách liên kết hai chiều có dạng:





III.2 Khai báo :

{ khai báo dữ liệu }

```
typedef struct node *nodep;
typedef int infotype;
struct node {
          infotype data;
          nodep next, pred;
         };
nodep F, L;
```

Tạo DSLK hai chiều rỗng:

```
void init(nodep *F,nodep *L)
{
  *F=NULL; *L=NULL;
}
```

III.3 Tạo DSLK hai chiều :

*** Giải thuật :**

- Tạo DS rỗng
- Lặp lại cho đến khi thôi
  - .Tạo một con trỏ p và nhập dữ liệu cho p
  - . Gán p^.pred = p^.next = NULL ;
  - . Nối p vào DS

*** Cài đặt :**

```
void taods2chieu(nodep *F, nodep *L)
{
 nodep p;
 char tl;
 randomize();
 printf("\n Bam fim bat ky de nhap, Bam ESC de thoi");
 do {
   p=(nodep)malloc(sizeof(struct node));
   p->data=random(100); printf("%4d",p->data);
   p->next=p->pred=NULL;
   if (*F==NULL)  *F=p;
   else   {  (*L)->next=p; p->pred=*L; }
   *L=p;
   tl=getch();
  } while (tl!=27); printf("\n");
}
```

III.4 Duyệt DSLK :

*** Giải thuật :**

- Gán p = F ;
- Lặp lại cho đến khi p = NULL
  - . <Xử lý p->data>
  - . p = p^.next ;

*** Ví dụ :**

- . Chương trình Xem thuận hoặc Xem ngược
- . Chương trình tính tổng các phần tử trong DS số nguyên
- . Chương trình in ra các bản ghi thỏa điều kiện nào đó

*** Cài đăt :**

```
void xemthuan(nodep F)
{ nodep p;
  p=F;
  while (p)
  {
```

```
   printf("%4d",p->data);
   p=p->next;
  }
 printf("\n");getch();
}
void xemnguoc(nodep L)
{ nodep p;
  p=L;
  while (p)
  {
   printf("%4d",p->data);
   p=p->pred;
  }
 printf("\n");getch();
}
```

III.5 Chèn một phần tử mới vào DS tại vị trí thứ k :

*** Giải thuật :**

- Tm hai con trỏ tp và p lần lượt trỏ vào phần tử thứ k-1 vă thứ k
- Nếu không chèn được (i!=k) thì thông báo " Không chèn được "
- Ngược lai nếu chèn được thì

.Tạo con trỏ q và nhập dữ liệu cho q
. Gân q->pred = q->next = NULL
. nếu (k = = 1 0 thì (1)
. ngược lại nếu (p = =NULL) thì (2)
. ngược lại thì (3)

(1) : q->next = F; F->pred = q; F = q;
(2) : q->pred=L; q->next=NULL; L->next=q; L=q;
(3) : q->pred=tp; q->next=tp->next ; tp->next=q; p->pred=q;

*** Căi đặt :**

```
void chenk(infotype e,int k, nodep *F,nodep *L)
{
 nodep q,tp,p;
 int i=1;
 p=*F;
 while(p!=NULL && i<k)
 {
  i++;
  tp=p;
  p=p->next;
 }
 if (i==k) /*chèn được */
 {
  q=(nodep)malloc(sizeof(struct node));
  q->data=e;
  if (k==1)
```

```
  {
   q->pred=NULL;
   q->next=*F;
   (*F)->pred=q;
   *F=q;
 }
 else
  if (p==NULL)
  {
   q->next=NULL;
   q->pred=*L;
   (*L)->next=q;
   *L=p;
  }
  else
  {
   q->pred=tp;
   q->next=tp->next;
   tp->next=q;
   p->pred=q;
  }
 }
 else printf("\n Khong chen duoc !");
 getch();
}
```

III.6 Xoá phần tử thứ k trong DS :

*** Giải thuật :**

- Tìm hai con trỏ tp và p lần lượt trỏ vào nút thứ k-1 và thứ k
- Nếu không xóa được (p = = NULL) thì thông báo " Không xoá được "
- Ngược lại nếu xóa được thì
  - . Nếu (k = = 1) thì (1)
  - . ngược lại nếu (p = = pc) thì (2)
  - . ngược lại thì (3)
  - (1) : F=p->next ; F->pred=NULL ;
  - (2) : L=p->pred ; L->next=NULL ;
  - (3) : q= p->next ; tp->next = q ; q->pred = p->pred ;

*** Căi đặt :**

```
void xoak(int k,nodep *F,nodep *L)
{
 nodep p,q,tp; int i=1;
 if (k<1) exit(1);
 p=*F;
 if (k==1)
 {
  *F=p->next;
```

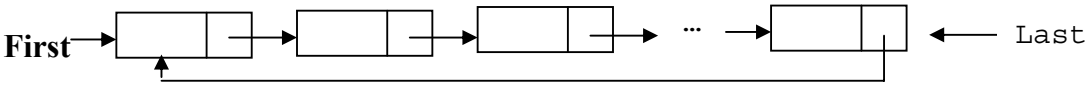
```
  (*F)->pred=NULL;
 }
 else
 {
   while(p!=NULL && i<k)
   {
    i++;
    tp=p;
    p=p->next;
   }

   if (p->next==NULL)
   {
    *L=p->pred;
    (*L)->next=NULL;
   }
   else
   {
    q=p->next;
    tp->next=q;
    q->pred=p->pred;
   }
  }
  free(p);
 }
```

IV. Danh sách liên kết vòng:

IV.1 Định nghĩa:

Danh sách liên kết vòng là một danh sách liên kết mà phần tử cuối cùng (Last) chỉ vào phần tử đầu tiên (First) của danh sách.



IV.2 Các phép toán trên danh sách:

Các phép toán trên danh sách liên kết vòng cũng tương tự với các phép toán trên danh sách đơn. Tuy nhiên khi sử dụng các giải thuật của danh sách liên kết đơn ta cần phải chú ý là vùng liên kết của phần tử cuối cùng chỉ vào phần tử đầu tiên của danh sách mà không chỉ vào NULL. Phần tử đầu First bây giờ không còn có ý nghĩa thật sự và nó có thể chỉ dùng để truy nhập vào một phần tử bất kỳ của danh sách liên kết vòng mà thôi. Danh sách liên kết vòng thích hợp với các phép tách và phép ghép danh sách.

Chương 4 NGĂN XẾP VÀ HÀNG ĐỢI

A. Ngăn xếp (Stack) :

I .Khái niệm :

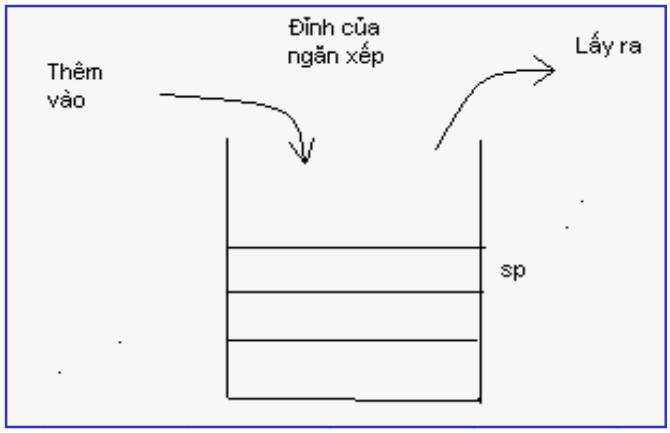
Ngăn xếp là 1 DS đặc biệt mà phép bổ sung và loại bỏ chỉ có thể tiến hành ở đầu DS. Trong trường hợp này đầu DS sẽ được gọi là đỉnh của ngăn xếp và ngăn xếp hoàn toàn được xác định bởi PT đỉnh này.

* Chú ý :

+ Trong lập trình ngăn xếp được coi là là 1 CTDL trừu tượng do đó khi sử dụng ngăn xếp người lập trình không cần biết đến cấu trúc bên trong của ngăn xếp mà chỉ quan tâm đến các phép toán có thể tiến hành đối với ngăn xếp.

+ Đối với ngăn xếp những PT được đưa vào đầu tiên thì sẽ được lấy ra sau cùng và ngược lại. Do vậy ngăn xếp còn có tên gọi là kiểu LIFO (Last In First Out)

II. Các phép toán trên ngăn xếp dùng mảng:



II.1 Khai báo :

```
#define max 100
int s[max+1];
int top;
```

II.2 Khởi tạo :

```
void initS()
{
 top=-1;
}
```

II.3 Kiểm tra ngăn xếp rỗng:

```
int emptyS()
{
 return (top==-1);
}
```

II.4 Kiểm tra ngăn xếp đầy:

```
int full()
{
 return (top==max);
}
```

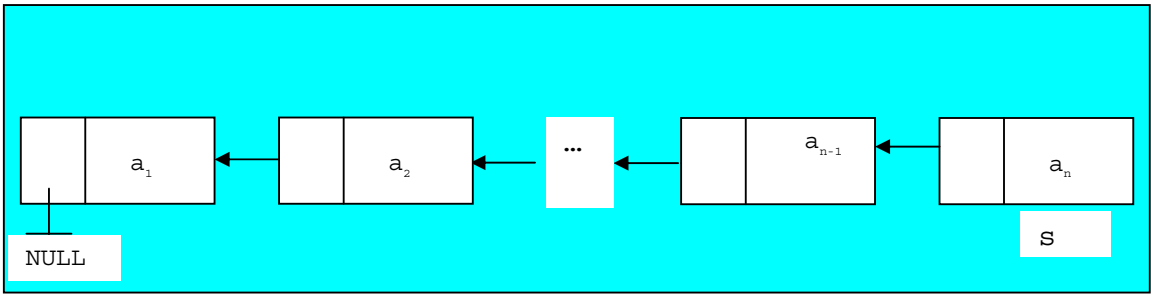
II.5 Thêm phần tử vào ngăn xếp : (điều kiện là NX chưa đầy)

```
int push(int value)
{
 if (top<max)
   s[++top]=value;
 return value;
}
```

II.6 Loại bỏ PT khỏi NX và đưa vào biến value :

```
int pop(int *value)
{
 *value=s[top--];
 return (*value);
}
```

III. Các phép toán đối với ngăn xếp dùng danh sách liên kết :



III.1 Khai báo dữ liệu kiểu NX :

```
typedef struct node *nodep;
typedef int infotype;
struct node {
  infotype data;
  nodep next;
  };
typedef nodep stack;
stack s;
```

III.2 Tạo 1 NX rỗng :

```
stack initS()
{
 stack s;
 s=(stack) malloc(sizeof(struct node));
 if (s==NULL)
  {
```

```
   printf("\n Khong du bo nho");
   exit(1);
  }
 else
   s->next=NULL;
 return s;
}
```

III.3 Kiểm tra tính rỗng của NX :

Hàm trả về đúng nếu NX rỗng, sai nếu ngược lại

```
int emptyS(stack s)
{
 return (s->next==NULL);
}
```

III.4 Đẩy 1 PT vào ngăn xếp : Bổ sung PT có giá trị vào đỉnh của NX

```
void push(infotype e, stack s)
{
 nodep tmp;
 tmp=(nodep) malloc(sizeof(struct node));
 if (tmp!=NULL)
 {
  tmp->data=e;
  tmp->next=s->next;
  s->next=tmp;
 }
}
```

III.5 Lấy 1 PT khỏi ngăn xếp : Loại bỏ 1 PT khỏi đỉnh NX và lấy giá trị của PT này vào biến t.

```
void pop(infotype *t,stack s)
{
 nodep cell;
 if (!emptyS(s))
  {
   cell=s->next;
   *t=cell->data;
   s->next= cell->next;
  }
}
```

III.6 Xem giá trị của PT ở đỉnh NX : tương tự như hàm Pop nhưng PT ở đỉnh không bị loại bỏ khỏi NX

```
void peep(infotype *t,stack s)
{
 nodep cell;
 if (!emptyS(s))
  {
```

```
    cell=s->next;
    *t=cell->data;
   }
}
```

IV. Ứng dụng của ngăn xếp :

IV.1 Bài toán đổi 1 số nguyên dương cơ số 10 sang cơ số 2

*) GT : - nhập n, khởi động NX rỗng
- lặp lại
+ chia n cho 2 và đưa phần dư vào NX
+ đặt n = n/2
cho đến khi n = 0
- Dở các PT khỏi NX và in ra màn hình

*) Cài đặt :

```
void chuyendoi(int s)
{
 int r;
 printf("\n Dang nhi phan cua %d la:" ,s);
 initS();
 while (s>=1)
 {
  r=s%2;
  push(r);
  s/=2;
 }
 while (!emptyS())
  {
   pop(&r);
   printf("%d",r);
  }
 getch();
}
```

IV.2 Tính giá trị của biểu thức hậu tố :

a) Các khái niệm :

- Giả sử t là phép toán 2 ngôi thì có 3 cách để thể hiện thao tác áp đặt phép toán t trên 2 toán hạng a, b
- Dùng ký pháp trung tố (phép toán ở giữa) : a t b
- tiền tố : t a b
- hậu tố : a b t

Trong đó ký pháp hậu tố còn được gọi là ký pháp nghịch đảo Ba Lan và tương ứng với 3 ký pháp ở trên ta sẽ có 3 biểu thức khác nhau : bt trung tố tiền tố hậu tố

Ví dụ :

Biểu thức trung tố	Biểu thức hậu tố
5 + 3	5 3 +
3 + 7 + 9	3 7 + 9 +

3 + 7 * 9	3 7 9 * +
3 * (7 + 9)	3 7 9 + *

b) Giải thuật tính giá trị biểu thức hậu tố :

(1) Khởi động NX S

(2) Lặp lại khi chưa đọc hết biểu thức

- Đọc 1 PT của biểu thức vào x
- nếu x là toán hạng thì đẩy x vào NX
- ngược lại :
  - . dở b khỏi NX
  - . dở a khỏi NX
  - . đẩy a x b vào NX

(3) Nếu trong NX còn 1 PT thì đó là kết quả

* Chú ý : Nếu ở bước dở a hoặc dở b ngăn xếp bị rỗng (cạn) hoặc nếu ở bước (3) NX có nhiều hơn 1 PT thì chứng tỏ biểu thức hậu tố bị sai

c) Cài đặt:

```
int tinh(char x,int a,int b)
{
 int t;
 switch(x)
 {
 case '+':
   t=a+b;
   break;
 case '-':
   t=a-b;
   break;
 case '*':
   t=a*b;
   break;
 case '/':
   t=a/b;
   break;
 }
 return t;
}
void gtbtht(char st[])
{
 int a,b,ok=1,i=0,l=strlen(st);
 char x;
 initS();
 while (i<=l && ok)
 {
  x=st[i];
  if (x >='0' && x<='9')
     push(x-48);
```

```
  else if (x>=42 && x<=47)
  {
    pop(&b);
    if (emptyS())  ok=0;
    else
    {
     pop(&a);
     push(tinh(x,a,b));
    }
   }
   i++;
  }
  if (!ok) printf("\n Bieu thuc du toan tu");
  else
  {
   pop(&a);
   if (emptyS()) printf("\n Gia tri la: %d",a);
   else    printf("\n Bieu thuc du toan hang");
  }
 getch();
}
```

IV.3 Chuyển biểu thức trung tố thành hậu tố :(tự làm dùng ngăn xếp và hàng đợi)

*** Giải thuật :**

+ Khởi tạo ngăn xếp rỗng
+ Lặp lại các bước sau cho đến khi hết biểu thức hoặc gặp lỗi
  (a) Đọc 1 phần tử của biểu thức
  (b) Nếu nó là :
    (i) dấu '(' thì đẩy vào ngăn xếp
    (ii) dấu ')' thì lấy ra và hiển thị các phần tử của ngăn xếp cho đến khi gặp dấu '(' . Nếu không gặp dấu '(' mà ngăn xếp rỗng thì biểu thức có lỗi.
    (iii) Toán tử : lặp lại quá trình sau cho đến khi đưa được toán tử vào ngăn xếp :
      - Nếu ngăn xếp rỗng hay toán tử được ưu tiên hơn phần tử ở đỉnh NX thì đẩy toán tử vào NX.
      - Ngược lại lấy phần tử khỏi NX và hiển thị nó (dấu '(' có độ ưu tiên bé hơn các toán tử)
    (iv) Toán hạng : hiển thị nó
+ Chừng nào ngăn xếp còn chưa rỗng thì lấy các phần tử ra khỏi NX và hiển thị chúng.

*** Cài đặt :**

// Program chuyen_bt_trungto_sang_hauto;

```
typedef struct node1 *stack1;
struct node1 {
  char data;
```

```
  stack1 next;
  };
stack1 initS1()
{
 stack1 s;
 s=(stack1) malloc(sizeof(struct node1));
 if (s==NULL)
  {
   printf("\n Khong du bo nho");
   exit(1);
  }
 else
   s->next=NULL;
 return s;
}
int emptyS1(stack1 s)
{
 return (s->next==NULL);
}
void push1(infotype1 e, stack1 s)
{
 stack1 tmp;
 tmp=(stack1) malloc(sizeof(struct node));
 if (tmp!=NULL)
 {
  tmp->data=e;
  tmp->next=s->next;
  s->next=tmp;
 }
}
void pop1(infotype1 *t,stack1 s)
{
 stack1 cell;
 if (!emptyS1(s))
  {
   cell=s->next;
   *t=cell->data;
   s->next= cell->next;
  }
}
int douutien(char ch)
{
 int t;
 switch(ch)
 {
  case '(': t=1;break;
```

```
  case '+': t=2;break;
  case '-': t=2;break;
  case '*': t=3;break;
  case '/': t=3;break;
 }
 return t;
}
void chuyen(char *tt,char **ht)
{
 stack1 s;
 int error=0,ok,i=0,l=strlen(tt);
 char ptkt,ptbt;
 char *c="t";
 s=initS1();
 strcpy(*ht,"");
 while (i<l && !error)
 {
  ptkt=tt[i];
  if (ptkt=='(') push1(ptkt,s);
  else if (ptkt==')')
   {
    ok=0;
    do {
      if (emptyS1(s)) error=1;
      else
       {
         pop1(&ptbt,s);
         if (ptbt!='(') {*c=ptbt;strcat(*ht,c);}
         else ok=1;
       }
    }while (!ok && !error);
  }
  else if (ptkt>=42 && ptkt<=47)
   {
    ok=0;
    while (!emptyS1(s) && !ok)
    {
     pop1(&ptbt,s);
     if (douutien(ptkt)<=douutien(ptbt))
       { *c=ptbt;strcat(*ht,c); }
     else
       {
     push1(ptbt,s);
     ok=1;
       }
    }
```

```
      push1(ptkt,s);
    }
   else
     if(ptkt>='0' && ptkt<='9') { *c=ptkt;strcat(*ht,c); }
   i++;
  }
  while (!emptyS1(s) && !error)
  {
   pop1(&ptbt,s);
   if (ptbt != '(') { *c=ptbt;strcat(*ht,c); }
   else error=1;
  }

 if (error) *ht="Co loi trong bt trung to";
}
```

B. Hàng đợi (Queue) :

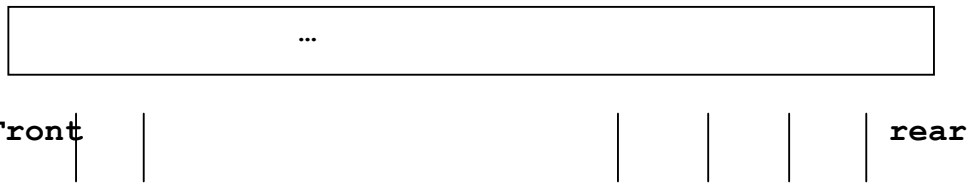
V. Khái niệm :

Hàng đợi (HĐ) là 1 DS đặc biệt mà phép bổ sung PT chỉ có thể tiến hành ở 1 đầu DS (đầu này gọi là lối vào của hàng đợi) còn phép loại bỏ PT chỉ có thể tiến hành ở đầu còn lại của HĐ (đầu này gọi là lối ra của HĐ)

* Chú ý : HĐ cũng có thể coi là 1 CTDL trừu tượng nên người lập trình không cần quan tâm đến cấu trúc bên trong của HĐ mà chỉ có thể biết các phép toán đối với HĐ Đối với HĐ những PT được đưa vào đầu tiên sẽ được lấy ra đầu tiên do đó kiểu HĐ là kiểu FIFO (First In First Out)

VI. Các phép toán trên hàng đợi dùng mảng :

VI.1 Khai báo :




```
#define max 100
int q[max + 1];
int front,rear,size;
```

* Khắc phục hàng bị tràn : Hàng bị đầy khi tất cả các ô đã được sử dụng. Hàng bị tràn là còn ô trống nhưng không thêm PT mới được vì rear đã tịnh tiến đến cuối hàng. Để khắc phục hàng bị tràn ta sử dụng qui tắc sau :

+ Qui tắc di chuyển tịnh tiến :
  - Khi thêm vào hàng trống thì thêm vào ô thứ nhất của hàng
  - Khi thêm vào hàng đang có rear = n thì dồn hàng lên đầu

+ Qui tắc di chuyển vòng :

- Khi thêm vào hàng trống hoặc đang có rear = n thì thêm vào ô thứ nhất của hàng

VI.2 Khởi tạo hàng rỗng:

```
void initq()
{
 front=rear=size=0;
}
```

VI.3 Kiểm tra tính rỗng và tính đầy của hàng đợi :

```
int emptyq()  // rỗng
{
 return (size==0) ;
}

int full()         // đầy
{
 return (size==max);
}
```

VI.4 Thêm phần tử vào hàng :

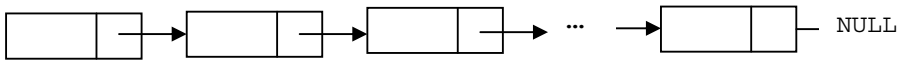
```
int push(int value)
{
 if (size<max)
  {
   size++;
   q[rear++]=value;
   if (rear==max)
     rear=0;
  }
   return value;
}
```

VI.5 Loại bỏ phần tử khỏi hàng :

```
int pop(int *value)
{
 if (size>0)
 {
  *value=q[front++];
  if (front>max) front=0;
  size--;
 }
 return *value;
}
```

VII Các phép toán trên hàng đợi dùng danh sách liên kết :

VII.1 Khai báo HĐ :



```
       Front                                    rear

typedef struct node *nodep;
typedef int infotype;
struct node {
          infotype data;
          nodep next;
         };
struct queuerecord {
          nodep front,rear;
          };
typedef struct queuerecord queue;
```

VII.2 Tạo hàng đợi rỗng :

```
void init(queue *q)
{
 (*q).front=NULL;
 (*q).rear=NULL;
}
```

VII.3 Kiểm tra tính rỗng của hàng đợi :

```
int empty(queue q)
{
  return (q.front==NULL);
}
```

VII.4 Đưa 1 PT vào hàng đợi :

```
void push(infotype NE,queue *q)
{
 nodep p;
 p=(nodep)malloc(sizeof(struct node));
 p->data=NE;
 p->next=NULL;
 if ((*q).front ==NULL) (*q).front=p;
 else ((*q).rear)->next=p;
 (*q).rear=p;
}
```

VII.5 Lấy 1 PT ra khỏi hàng đợi :

```
void pop(infotype *DE,queue *q)
{
 nodep p;
 if (!empty(*q))
 {
  p=(*q).front;
  *DE=p->data;
  (*q).front=p->next;
  free(p);
```

```
  }
}
```

VIII. Ưng dụng của hàng đợi :

+ Trong quá trình chuyển biểu thức trung tố sang biểu thức hậu tố ta có thể sử dụng hàng đợi để lưu trữ biểu thức hậu tố

+ Trong bộ đếm dùng trong xử lý tập tin cũng có thể được cài đặt bằng cách dùng hàng đợi

+ Minh họa nội dung của NX sau mỗi (lần) khi thực hiện phép toán thuộc dãy :

E X * A * * M P L * * E * *

Trong đó mỗi chữ cái đại diện cho phép toán đẩy chữ cái đó vào NX còn dấu * đại diện cho phép toán dở PT ra khỏi NX { $ ngăn xếp rỗng }

Dùng Ngăn xếp		Dùng Hàng đợi	
Phép toán	Ngăn xếp	Phép toán	Hàng đợi
E	$ E		$
X	$ E X	E	E $
*	$ E	X	X E $
A	$ E A	*	X $
*	$ E	A	A X $
*	$	*	A $
M	$ M	*	$
P	$ M P	M	M $
L	$ M P L	P	P M *
*	$ M P	L	L P M $
*	$ M	*	L P $
E	$ M E	*	L $
*	$ M	E	E L $
*	$	*	E $

VIII.1 Sắp xếp danh sách bằng phương pháp dùng hàng đợi (phương pháp RadixSort)

*** Giải thuật :**

(1) Tạo 10 hàng đợi rỗng từ Q[0] đến Q[9]

(2) Tìm m là số chữ số của phần tử lớn nhất của danh sách

(3) for (i = 1; i <= m; i ++)

(a) Thực hiện các bước sau cho đến khi hết danh sách

- Lấy phần tử x từ danh sách
- Kí hiệu k = chữ số thứ i của x, từ phải qua
- Đưa phần tử x vào Q[k]

(b) for (j = 0; j < 10; j ++)

Đưa tất cả các phần tử từ Q[j] vào lại trong danh sách.

Kết quả là các phần tử trong danh sách tăng dần.

Minh họa : Giả sử ta có mảng số nguyên như sau :

317 249 5 21 3 127 468 52

- Tạo 10 hàng đợi rỗng Q_0 -> Q_9
- Tính m = 3 { là chiều dài của số 468 }

Q0		3 5	3 5 21 52
Q1	21	317	127
Q2	52	21 127	249
Q3	3		317
Q4		249	468
Q5	5	52	
Q6		468	
Q7	127 317		
Q8	468		
Q9	249		

```
* i = 1
(a) ••a vào hàng ••i các
pt c•a m•ng
(b) M1 : 21  52  3  5
317  127  468  249
* i = 2
(a) ••a vào hàng ••i các
pt c•a m•ng
(b) M2 : 3  5  317  21
127  249  52  468
* i = 3
(a) ••a vào hàng ••i các
pt c•a m•ng
(b) M3 : 3  5  21  52
127  249  317  468
```

*** Cài đặt:**

```
void init10(mangq mq)
{
 int i;
 for (i=0;i<10;i++)
  init(&mq[i]);
}
int maxx(int *m,int n)
{
 int i,t=m[0];
 for(i=1;i<n;i++)
  if (t<m[i]) t=m[i];
 return t;
}

int scs(int n)
{
 int t=0;
 while (n>0)
 {
  t++;
  n=n/10;
 }
 return t;
}
void radixsort(int *A,int n)
{
 int m,ii,i,j,k,x;
 init10(mq);
```

```
 m=scs(maxx(A,n));
 printf("m=%d",m);getch();
 for (i=0;i<m;i++)
 {
  for (j=0;j<n;j++)
  {
   k=(A[j]/(int)pow(10,i))%10;
   push(A[j],&mq[k]);
   }
  ii=0;
  for (j=0;j<10;j++)
  {
   while (!empty(mq[j]))
   {
    pop(&x,&mq[j]);
    A[ii++]=x;
   }
  }
 }
}
```

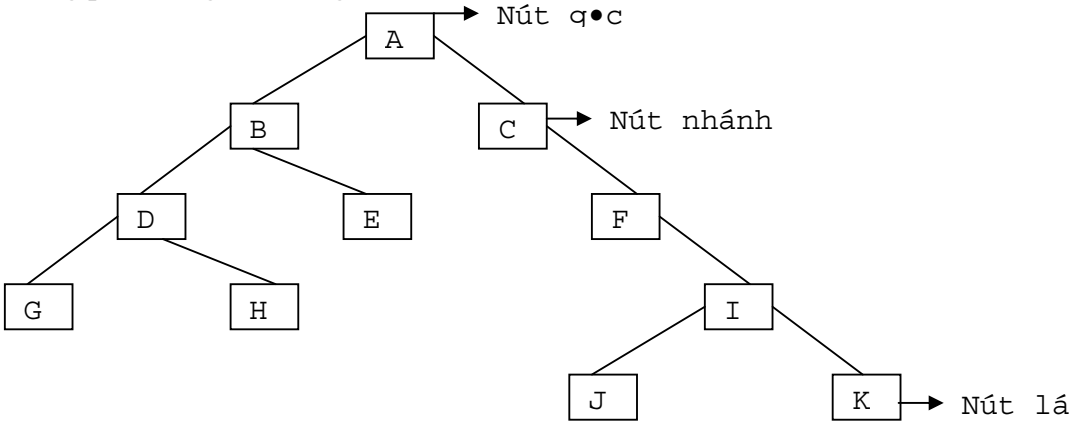
VIII.2 Phân tích một số nguyên dương >1 thành các thừa số nguyên tố:

```
void phantich(int n)
{
 queue q;
 int i=2;
 init(&q);
 printf("%d = ",n);
 while (n>1)
 {
  if (n%i == 0)
   {
    push(i,&q);
    n=n/i;
   }
  else  i++;
 }
 while (!empty(q))
 {
  pop(&i,&q);
  if (!empty(q)) printf("%d*",i);
  else  printf("%d",i);
 }
 getch();
}
```

Chương 5 **CÂY**

I. Các khái niệm :

+ Cây là 1 tập hợp gồm các phần tử (PT) có cùng kiểu dữ liệu, mỗi PT được gọi là 1 nút. Một nút đứng riêng được gọi là 1 cây và nút đó cũng là nút gốc của cây
+ Nếu ta có cây T_1, T_2, T_3,..., T_k với các gốc lần lượt là t_1, t_2, t_3,..., t_k và ta có 1 nút t không thuộc các t kể trên thì ta có thể lập 1 cây mới bằng cách liên kết nút t với t_1, t_2, t_3,..., t_k . Cây mới sẽ có gốc là t.
+ Chấp nhận 1 cây đặc biệt là cây rỗng tức là cây không có nút nào
+ Nút cha - nút con : nếu có một nút t được liên kết với các nút t_1, t_2, t_3,..., t_k thì ta gọi t là nút cha còn t_1, t_2, t_3,..., t_k là nút con.
+ Nút lá - nút nhánh : một nút không có nút con gọi là nút lá, một nút không phải nút lá, không phải nút gốc được gọi là nút nhánh



+ Bậc : là số lượng nút con của nút đó. Bậc lớn nhất của cây gọi là bậc của cây
+ Mức : Nút gốc có mức là 1. Nếu 1 nút có mức là i thì các con của nó sẽ có mức là i +1
+ Chiều cao (độ sâu) : Mức lớn nhất trong cây được gọi là chiều cao của cây
+ Cây có thứ tự : nếu trong 1 cây ta quan tâm đến thứ tự của các nút con của 1 nút bất kỳ thì ta gọi là cây có thứ tự, ngược lại thì cây không có thứ tự
+ Nếu có 1 tập hữu hạn các cây phân biệt thì ta gọi tập đó là rừng

II. Cây nhị phân (Binary tree) :

II.1 Định nghĩa : Cây nhị phân là cây bậc hai, có thứ tự (mỗi nút có tối đa 2 nút con)

II.2 Biểu diễn :

```
typedef int infotype;
typedef struct node *tree;
struct node {
     infotype data;
     tree left,right;
     } ;
Tree T;
```

*** Chú ý :**

- Tương tự như mảng và danh sách liên kết phần dữ liệu của 1 nút có thể khá phức tạp nhưng ta tinh giản xuống còn 1 số nguyên, số nguyên này còn gọi là khóa của 1 nút
- Con trỏ left, right dùng để trỏ tới nút con bên trái và nút con bên phải. Tuy nhiên left, right cũng đồng thời là con trỏ trỏ đến cây con trái và phải. Như vậy có thể coi left và right là cây con bên trái và cây con bên phải.

II.3 Duyệt cây :

- Nội dung duyệt cây gồm 3 việc chính :

Nếu như cây tồn tại tức là cây khác rỗng, thực hiện các việc sau:

+ thăm nút gốc
+ duyệt cây con bên trái
+ duyệt cây con bên phải

* 3 giải thuật để duyệt cây :

+ Duyệt theo thứ tự trước (Preorder) : thăm gốc --> duyệt cây trái --> duyệt cây phải

+ Duyệt theo thứ tự giữa (Inorder) : duyệt cây trái --> thăm gốc --> duyệt cây phải

+ Duyệt theo thứ tự sau (Postorder) : duyệt cây trái --> duyệt cây phải --> thăm gốc

*** Cài đặt thủ tục duyệt giữa (đệ qui) :**

```
void inorder(Tree T)
 {
  if (T)
  {
   inorder(T->left);
   printf("%4d",T->data);
   inorder(T->right);
  }
 }
```

*** Giải thuật duyệt cây T theo thứ tự trước không đệ qui :**

+ Khởi tạo ngăn xếp rỗng
+ Đặt p = T ; Ok = 1 ;
+ Lặp lại khi (p !=NULL và Ok)
  - <xử lý p->data>
  - nếu (p->right !=NULL) thì đẩy p->right vào NX
  - nếu (p->left != NULL) thì p = p->left
  - ngược lại thì
    . nếu NX rỗng thì Ok = 0
    . ngược lại thì lấy 1 phần tử từ NX và gọi là p

*** Giải thuật duyệt cây T theo thứ tự giữa không đệ qui :**

+ Khởi tạo NX rỗng
+ Đặt p = T ;

+ Lặp lại
  - chừng nào (p!=NULL) thì
    . đẩy p vào NX
    . p = p->left ;
  - nếu NX không rỗng thì
    . lấy 1 phần tử từ NX gọi là p
    . <xử lý p->data>
    . p = p->right
    . Ok = true
  - ngược lại thì Ok = False
+ cho đến khi Ok = False

*** Cài đặt duyệt cây T theo thứ tự giữa không đệ qui :**

```
void Inorder_khongDQ(Tree T)
{
 Tree p;
 inits(s);
 p = T;
 do
 {
   while (p!=NULL)
     {
        push(p);
        p = p->left;
     }
   if (!emptys(s))
     {
       pop(&p);

     printf("%4d",p->data);
     p=p->right;
     ok =1;
     }
   else  ok = 0;
 } while (ok);
}
```

*** Giải thuật duyệt cây T theo mức :**

(1) Tạo hàng đợi rỗng
(2) Đưa gốc cây T vào hàng đợi
(3) Lặp lại khi HĐ chưa rỗng
  (a) lấy 1 phần tử x từ hàng đợi
  (b) Thăm x
  (c) Đưa các con của x vào hàng đợi

- **Cài đặt duyệt cây T theo mức :**

```
void level(Tree T)
{
```

```
 Tree p;
 initq();
 if (T!=NULL) push(T);
 while (!emptyq())
 {
  pop(&p);
  printf("%4d",p->data);
  if (p->left != NULL) push(p->left);
  if (p->right != NULL) push(p->right);
 }
}
```

*** Xây dựng hàm tìm số nút của 1 cây :**

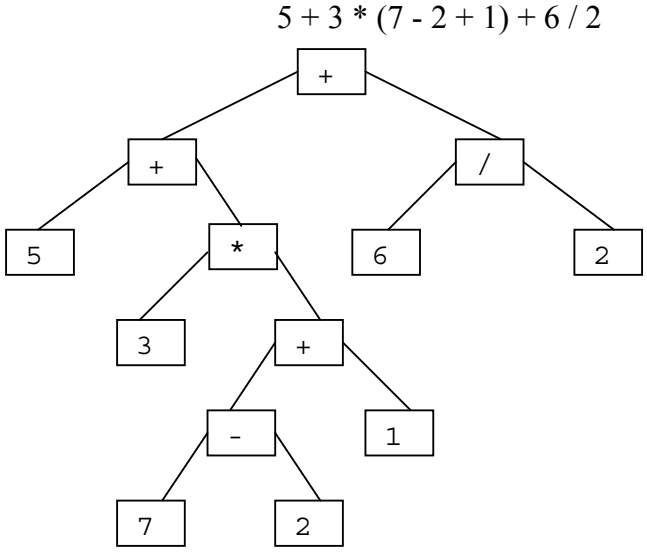
```
int     sonut( Tree t )
{
        If  (t = = NULL)    return 0 ;
        Else  return(1 + sonut(t->left) + sonut(t->right)) ;
}
```

***Minh họa quá trình duyệt cây (xem hình trang 42):**

Trình tự các nút được thăm :

- Theo trình tự Preorder : A B D G H E C F I J K
- Theo trình tự Inorder : G D H B E A F J I K C
- Theo trình tự Postorder : G H D E B J K I F C A

(ii) Biểu thức số học có thể biểu diễn ở dạng cây nhi phân :

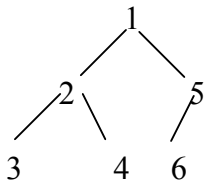


Trình tự các nút được thăm :

- Preorder : + + 5 * 3 + - 7 2 1 / 6 2 { biểu thức tiền tố }
- Inorder : 5 + 3 * 7 - 2 + 1 + 6 / 2 { biểu thức trung tố }
- Postorder : 5 3 7 2 - 1 + * + 6 2 / + { biểu thức hậu tố }

III. Cây cân bằng hoàn toàn :

III.1 Khái niệm : Một cây được gọi là cây cân bằng hoàn toàn nếu với mọi nút của cây, số lượng nút của cây con trái và cây con phải chênh nhau không qúa 1. Ví dụ : Cây có 6 nút



III. 2 Lập cây cân bằng hoàn toàn có n nút :

Giả sử các giá trị khóa được nhập từ bàn phím

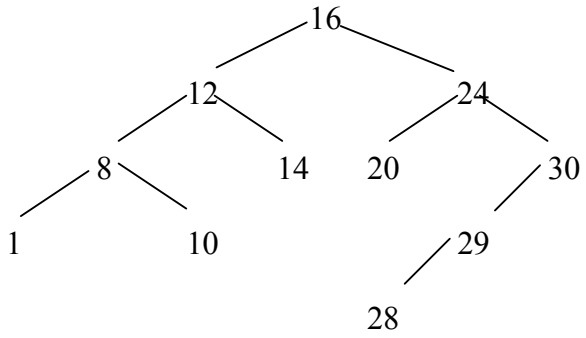
```
Tree    CayCBHT(int n)
{
  Tree  t ;
   if  (n = = 0)  return NULL;
   else
        {
              t = (Tree) malloc(sizeof(struct node) ;
              scanf("%d", &x);
              t->data = x ;
              t->left  =  CayCBHT(n div 2) ;
              t->right  :=  CayCBHT(n - 1 - n div 2) ;
              return  t ;
        }
}
```

IV. Cây tìm kiếm nhị phân (Binary Search Tree) :

IV.1 Khái niệm :

Cây tìm kiếm nhị phân là cây thỏa mãn điều kiện sau : với mọi nút của cây thì giá trị khóa của các nút cây con bên trái nhỏ hơn giá trị của nút đó và giá trị khóa của các cây con bên phải lớn hơn giá trị khóa của nút đó.

Ví dụ :



IV.2 Thêm một nút vào cây tìm kiếm nhị phân :

```
// Hàm chèn nút p (con trỏ p) vào cây T
void insert(Tree p,Tree *T)
{
 if (p->data<(*T)->data)
  if ((*T)->left)
   insert(p,&(*T)->left);
  else
   (*T)->left=p;
  else
   if ((*T)->right)
    insert(p,&(*T)->right);
   else
    (*T)->right=p;
}
// Hàm chèn một nút có dữ liệu là e vào cây T
void insert_node(infotype e,Tree *T)
{
 Tree p;
 p= (Tree)malloc(sizeof(struct node));
 p->data=e;
 p->left=NULL;
 p->right=NULL;
 if (*T ==NULL)
  (*T)=p;
 else
  insert(p,T);
}
```

IV. 3 Tạo cây tìm kiếm nhị phân:

Giải thuật:

- Khởi tạo cây rỗng T
- Lặp lại
  . nhập dữ liệu cho nút
  . chèn nút vào cây T
- Cho đến khi hết cây

Cài đặt:

```
void insert(infotype e,Tree *T)
{
 Tree p;
 p= (Tree)malloc(sizeof(struct node));
 p->data=e;
 p->left=NULL;
 p->right=NULL;
 if (*T ==NULL)
  (*T)=p;
```

```
 else
  if (e<(*T)->data) insert(e,&(*T)->left);
  else insert(e,&(*T)->right);
}

void taocay(Tree *T)
{
 infotype e;
 do {
  printf("\n nhap so nguyen, (-1 de thoi):");
  scanf("%d",&e);
  if (e!=-1)
   insert(e,&(*T));
  } while (e!=-1);
}
```

IV.4 Tìm kiếm một nút trong cây tìm kiếm nhị phân :

Bài toán: Giả sử ta có cây T. Tìm kiếm một nút có dữ liệu là se (search_element) trong cây này. Nếu tìm thấy biến found = 1 và nút p trỏ vào nút cần tìm (nút có dữ liệu là se), parent là cha của nút p. Nếu không tìm thấy se trong cây T thì biến found = 0.

Cài đặt:

```
void search(infotype se,Tree T, Tree *p,Tree *parent,int
*found)
{
 *p=T;*parent=NULL;*found= 0;
 while (*p!=NULL && !*found)
 {
  if ((*p)->data==se) *found=1;
  else
   {
    *parent=*p;
    if (se<(*p)->data) *p=(*p)->left;
    else *p=(*p)->right;
   }
 }
}
```

IV.5 Loại bỏ một nút khỏi cây tìm kiếm nhị phân :

Bài toán : Giả sử T là 1 cây tìm kiếm nhị phân và DelElement là 1 giá trị khóa. Cần loại bỏ nút có giá trị khóa = DelElement sao cho cây còn lại vẫn là cây tìm kiếm nhị phân.

Giải thuật :

Gọi p là nút muốn xoá (nút có dữ liệu là DelElement), parent là cha của nút muốn xoá p, child là con của nút muốn xoá (nếu có).

Có ba trường hợp:

1. Nếu nút muốn xoá p là nút lá:
   - Nếu (parent->left = = p) thì parent->left = NULL

- Nếu (parent->right = = p) thì parent->right = NULL

2. Ngược lại nếu nút muốn xoá p là nút chỉ có một nút con là child:
   - Nếu (parent->left = = p) thì parent->left = child
   - Nếu (parent->right = = p) thì parent->right = child
3. Ngược lại nếu nút muốn xoá p là nút có đầy đủ hai nút con:
   - tìm x là nút tận cùng bên phải của cây con bên trái của p, **hoặc** tìm x là nút tận cùng bên trái của cây con bên phải của p.
   - chép dữ liệu của x vào p (p->data = x->data)
   - Gán p = x ;
   - loại bỏ nút p (lúc này p thuộc trường hợp 1 hoặc 2)

Cài đặt :

```
void search(infotype e,Tree T, tree *p,Tree *parent,int
*found)
{
 *p=T;*parent=NULL;*found= 0;
 while (*p!=NULL && !*found)
 {
  if ((*p)->data==e) *found=1;
  else
   {
    *parent=*p;
    if (e<(*p)->data) *p=(*p)->left;
    else *p=(*p)->right;
   }
 }
}
void del(infotype e, Tree *T)
{
  int found;Tree p,parent,x,chil;
  search(e,&(*T),&p,&parent,&found);
  if (!found)
  {
   printf("\nKhong co %d trong cay",e);getch();
   exit(1);
  }
  if ((p->left) && (p->right))
  {
    x=p->right;
    if (x->left==NULL) parent=p;
    while (x->left)
    {
     parent=x;
     x=x->left;
    }
    p->data=x->data;
```

```
    p=x;
  }
    chil=p->left;
    if (chil==NULL) chil=p->right;
    if (parent==NULL) T=chil;
    else if (parent->left==p) parent->left=chil;
      else parent->right=chil;
    free(p);
}
```

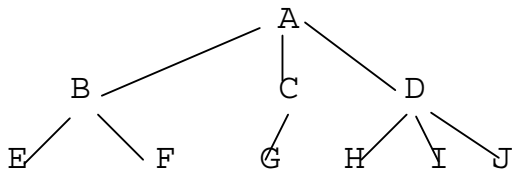
V. Cây tổng quát (nhiều nhánh) :

V.1 Biểu diễn cây tổng quát :

Một cây tổng quát cấp m có thể sử dụng cách biểu diễn móc nối tương tự như đối với cây nhị phân. Như vậy ứng với mỗi nút tất phải dành ra m trường móc nối để trỏ tới các con của nút đó và như vậy số "mối nối không " sẽ rất nhiều : nếu cây có n nút sẽ có tới n(m-1) +1 "mối nối không" trong số m . n mối nối.
Còn nếu tùy theo số con của từng nút mà định ra mối nối nghĩa là dùng nút có kích thước thay đổi thì sự tiết kiệm không gian nhớ này sẽ phải trả giá bằng những phức tạp của quá trình xử lý trên cây.

Một trong những phương pháp khá hiện thực là biểu diễn cây tổng quát bằng cây nhị phân. Như vậy quan hệ giữa các nút trên cây tổng quát chỉ được thể hiện qua hai đặc điểm thôi.

Ví d• : Xét cây • hình bên. V•i nút B : con c•c trái là E, em k• c•n ph•i là C.
V•i nút D : con c•c trái là H. em k• cân ph•i không có.
Nh• v•y n•u m•i nút có qui cách



Child	Info	Sibling

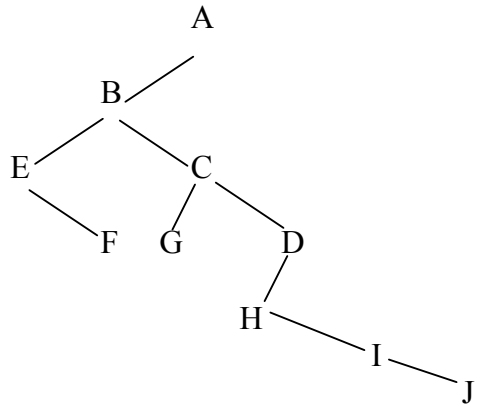
Ta nhận thấy, bất kỳ một nút nào trên cây tổng quát, nếu có thì chỉ có :

- một nút con cực trái (con cả)
- một nút em kế cận phải

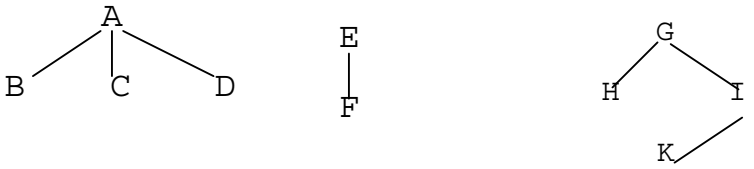
Lúc đó cây nhị phân biểu diễn cây tổng quát theo hai quan hệ này được gọi là *cây nhị phân tương đương*.

- Child : con trỏ, trỏ tới nút con cực trái
- Sibling : con trỏ, trỏ tới nút em kế cận phải

Vậy ta có thể biểu diễn cây tổng quát nêu trên bằng cây nhị phân tương đương như sau:



Ta thấy ngay là nối phải của nút gốc bao giờ cũng là mối nối không vì gốc không có em kế cận phải. Nhưng nếu xét một rừng thì tình trạng trên không xuất hiện. Vì vậy có thể biểu diễn rừng bằng một cây nhị phân tương đương (trường hợp một cây thì coi như rừng đặc biệt)
Ví dụ : Ta có rừng :



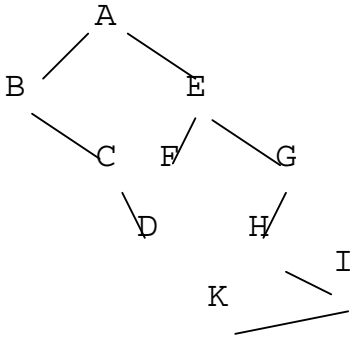
Có thể định nghĩa phép biến đổi tổng quát đối với rừng như sau : Nếu T_1, T_2, ..., T_n là một rừng thì cây nhị phân tương đương biểu diễn rừng đó kí hiệu bởi B(T_1, T_2, ..., T_n) sẽ là cây :

(1) rỗng, nếu n = 0

(2) có gốc là gốc của T_1, có cây con trái là B(T_{11},T_{12},..., T_{1m}) với T_{11},T_{12},..., T_{1m} là cây con gốc T_1, có cây con phải là B(T_2, ..., T_n)

V.2 Phép duyệt cây tổng quát :

Bi•u di•n r•ng b•ng cây nh• phân :



Phép duyệt cây tổng quát cũng được đặt ra tương tự hư đối với cây nhị phân nhưng cần phải xem xét thêm những điều sau đây :

(1) Sự nhất quán về thứ tự các nút được thăm giữa phép duyệt cây ấy (theo định nghĩa của phép duyệt cây tổng quát) và phép duyệt cây nhị phân tương đương của nó (theo định nghĩa của phép duyệt cây nhị phân) .

(2) Sự nhất quán giữa định nghĩa của phép duyệt cây tổng quát với định nghĩa của phép duyệt cây nhị phân. Vì cây nhị phân vẫn có thể được coi là cây, để duyệt theo phép duyệt cây tổng quát.

Nếu ta phỏng theo cách duyệt cây nhị phân thì ta sẽ xây dựng được định nghĩa của phép duyệt cây tổng quát T như sau :

*** Duyệt theo thứ tự trước :**

a) nếu T rỗng thì không làm gì

b) nếu T không rỗng thì :

(1) Thăm gốc của T

(2) Duyệt các cây con thứ nhất T_1 của gốc của T theo thứ tự trước

(3) Duyệt các cây con còn lại T2, T3,..., Tk của gốc T theo thứ tự trước

*** Duyệt theo thứ tự giữa :**

a) nếu T rỗng thì không làm gì

b) nếu T không rỗng thì :

(1) Duyệt các cây con thứ nhất T_1 của gốc của T theo thứ tự giữa

(2) Thăm gốc của T

(3) Duyệt các cây con còn lại T2, T3,..., Tk của gốc T theo thứ tự giữa

*** Duyệt theo thứ tự sau :**

a) nếu T rỗng thì không làm gì

b) nếu T không rỗng thì :

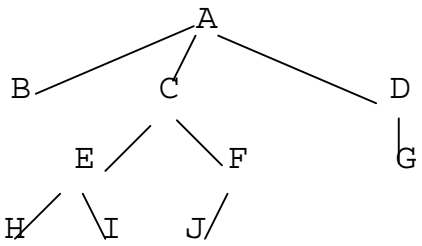
(1) Duyệt các cây con thứ nhất T_1 của gốc của T theo thứ tự sau

(2) Duyệt các cây con còn lại T2, T3,..., Tk của gốc T theo thứ tự sau

(3) Thăm gốc của T

Ví dụ :

Ta có cây :



thì dãy các nút được chọn là :

```
Th• t• TR••C :
 A  B  C  E  H  I  F  J  D  G
Th• t• GI•A :
 B  A  H  E  I  C  J  F  G  D
```

Th• t• SAU :
B H I E J F C G D A

Chương 7 Kỹ thuật băm

I. Các khái niệm cơ bản :

Giả sử ta có dãy khóa $x_1, x_2, ..., x_n$. Bài toán đặt ra là cần lưu trữ những phần tử này bằng cách nào đó để có thể tìm kiếm một cách nhanh nhất.

Nội dung :

- Dùng mảng gồm n phần tử để lưu trữ các khóa trên
- Xây dựng hàm h(x) (hàm băm)
- Lưu trữ khóa x tại vị trí h(x) của mảng nói trên (mảng đó gọi là bảng băm)
- Khi cần tìm kiếm 1 trị s nào đó ta cũng tiến hành bằng phương pháp trên tức là tìm s tại vị trí h(s)

Những vấn đề của kỹ thuật băm :

1) Giải quyết đụng độ : Đụng độ xảy ra khi 2 hoặc nhiều khóa có chung 1 giá trị băm, trong trường hợp này bảng băm chỉ lưu trữ được 1 khóa và ta phải tìm cách lưu trữ những khóa còn lại vào những vị trí khác

2) Tìm hàm băm h(x) thỏa mãn các điều kiện :
   - giảm đụng độ đến mức có thể
   - thời gian tính toán hợp lý

II. Xây dựng hàm băm h(x) :

II.1 Phương pháp chia : h(x) = x mod m (thông thường m là số nguyên tố)

II.2 Phương pháp nhân : - tìm x^2
- Trích ra 1 số chữ số ở giữa làm hàm h(x)

Ví dụ : x =125 $x^2 = 125 \times 125 = 15625$

h(x) = 56 hoặc 62

II.3 Phương pháp phân đoạn : thường áp dụng khi khóa có giá trị lớn

Nội dung : Cắt x thành các đoạn có độ dài bằng nhau rồi cộng lại lấy kết quả để làm hàm băm Ví dụ: x = 279 583 421 h(x) = 421 + 583 + 279 = 1283

III. Giải quyết đụng độ :

III.1 Phương pháp thử trực tiếp :

- Nếu h(x) = i và đã có phần tử tại vị trí thứ i thì ta tìm từ vị trí i +1 trở đi cho đến khi có vị trí trống (trong trường hợp đã tìm đến cuối bảng thì quay về vị trí đầu bảng tìm tiếp và nếu đến tại vị trí i thì kết luận bảng băm đã đầy) và điền x vào vị trí trống.

III. 2 Phương pháp kết nối (Phương pháp dây chuyền) :

Nội dung : Gắn kèm với mỗi PT bảng băm 1 danh sách liên kết để các giá trị khóa trùng giá trị băm.

Đây là chương trình làm việc trên bảng băm các số nguyên với hàm băm h(key)= key % 10.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#define TRUE 1
#define FALSE 0
```

```
#define M 10
struct nodes
{
int key;
struct nodes *next;
};
typedef struct nodes *NODEPTR;
NODEPTR bucket[M];  //mang cac con tro chi nut dau cua cac
bucket
//tac vu getnode(void)
NODEPTR getnode()
{
NODEPTR p;
p = (NODEPTR) malloc(sizeof(struct nodes));
return(p);
}
//Tac vu freenode: huy nut da cap phat
void freenode(NODEPTR p)
{
free(p);
}
// Ham bam
int hashfunc(int key)
{
return(key % M);
}
//Khoi dong cac bucket
void initbucket()
{
int b;
for (b=0;b <M; b++)
bucket[b] = NULL;
}
//Tac vu emptybucket;kiem tra but ket b co rong khong
int emptybucket (int b)
{
return(bucket[b] == NULL? 1:0);
}
//Tac vu push;them nut moi vao bucket thu b
void push(int x)
{
NODEPTR p;int b=hashfunc(x);
p = getnode();
p-> key = x;
p-> next =bucket[b];
bucket[b] = p;
```

```
}
//Tac vu remove :xoa nut co khoa k trong bang bam
void Remove(int k)
{
int b;
NODEPTR p, q;
b=hashfunc(k);
p=bucket[b];
while(p !=NULL && p->key !=k)
{
q=p;
p=p->next;
}
if (p == NULL)
printf("\n Khong co nut co khoa %d", k);
else  if (p==bucket[b]) bucket[b]=p->next;
   else  q->next=p->next;
free(p);
}
//Tac vu traversebucket:duyet bucket b
void traversebucket(int b)
{
NODEPTR p;
p=bucket[b];
while (p !=NULL)
{
printf("%3d",p->key);
p=p->next;
}
}
//Tac vu traverse:duyet bang ham
void traverse()
{
int b;
for (b=0;b<M; b++)
{
printf("\nBucket %d:",b);
traversebucket(b);
}
}
/*
Tac vu search: tim kiem mot nut trong bang bam, neu khong
tim thay ham nay tra ve tri 0, neu tim thay ham tra ve 1
*/

int search(int k,int *b)
```

```
{
NODEPTR p;

*b = hashfunc(k);
p = bucket[*b];
while(k!=p->key && p !=NULL) p = p->next;
if(p == NULL)  //khong tim thay
return 0;
else // tim thay
return 1;
}
// Chuong trinh chinh
void main()
{
int b,key,i,n,chucnang,c;
clrscr();
initbucket(); //khoi dong M bucket cua bang bam
do
{
     //Menu chinh cua chuong trinh
     printf("\n\Cac chuc nang cua chuong trinh:\n");
     printf("1:Them mot nut vao bang bam\n");
     printf("2:Them ngau nhien nhieu nut vao bang bam\n");
     printf("3: Xoa nut trong bang bam\n");
     printf("4: Xoa toan bo bang bam\n");
     printf("5: Duyet bang bam\n");
     printf("6: Tìm kiem tren bang bam\n");
     printf("0:Ket thuc chuong trinh\n");
     printf("\n Chuc nang ban chon:");
     scanf("%d",&chucnang);
switch(chucnang)
{
     case 1:
     {
          printf("\nTHEM MOT NUT VAO BANG BAM");
          printf("\n Khoa cua nut moi:");
          scanf("%d",&key);
          push(key);
          break;
     }
     case 2:
     {
          printf("\nTHEM NGAU HIEN NHIEU NUT VAO BANG
BAM");
          printf("\n Ban muon them bao nhieu nut:");
          scanf("%d",&n);
```

```
          for (i=0;i<n;i++)
          {
               key = random(100);
               push(key);
          }
          break;
     }
     case 3:
     {
          printf("\nXoa TREN BANG BAM");
          printf("\n khoa cua nut can xoa:");
          scanf("%d",&key);
          Remove(key);
          break;
     }
     case 4:
     {
          printf("\nXoa TOAN BO BANG BAM");
          printf("\nban co chac chan khong (c/k):");
          c=getch();
          if(c== 'C'|| c == 'c')
          initbucket()
          break;
     }
     case 5:
     {
          printf("\n DUYET BANG BAM");
          traverse();
          break;
     }
     case 6:
     {
          printf("\nTIM KIEM TREN BANG BAM");
          printf("\n Khoa can tim:");
          scanf("%d",&key);
          c=search(key,&b);
          if(c == 0)
          printf(" khong tim thay");
          else
          printf(" Tim thay trong bucket %d",b);
          getch();break;
     }
}
} while(chucnang!=0);
}
```

Chương 8 Sắp xếp và tìm kiếm ngoài

A. Sắp xếp ngoài :

I. Bài toán sắp xếp ngoài :

Phát biểu : Cho file có cấu trúc n bản ghi; Ta cần sắp xếp các bản ghi theo khóa nào đó. Nếu có thể nạp tất cả bản ghi vào 1 mảng nào đó thì có thể dùng một trong các phương pháp SX nội sắp xếp các PT của mảng, sau ddó ghi lại vào file. Tuy nhiên điều nầy không thể thực hiện được khi SX thứ tự tập tin lớn, vì bộ nhớ trong không thể chứa hết dữ liệu, vì thế cần có phương pháp thích hợp.

Phương pháp SX file thông dụng dựa trên ý tưởng thực hiện luân phiên hai công việc sau :

- tách tập tin lớn thành các (thường là 2) tập tin con có thứ tự nào đó
- trộn các tập tin con thành tập tin lớn có thứ tự

ví dụ :

II. Trộn 2 tập tin có thứ tự thành tập mới cũng có thứ tự :

Giải thuật :

(i) Mở file F1 và F2 để đọc , mở file F để ghi

(ii) Đọc PT đầu tiên của F1 vào biến x và PT đầu tiên của F2 vào biến y

(iii) Lặp lại các bước sau cho đến khi hết file F1 hoặc hết file F2

+ nếu x < y thì
  - ghi x vào file F
  - đọc PT tiếp theo của F1 vào x

+ ngược lai nếu x > y thì
  - ghi y vào file F
  - đọc PT tiếp theo của F2 vào y

+ ngược lại (x = y) thì
  - ghi x vào file F
  - đọc PT tiếp theo của F1 vào x
  - ghi y vào file F
  - đọc PT tiếp theo của F2 vào y

(iv) Nếu F1 chưa hết thì ghi các PT của F1 sang F, ngược lại thì ghi các PT của F2 sang F

ví dụ :

III. Phương pháp trộn tự nhiên :

Đây là phương pháp sắp xếp tập tin F, sử dụng 2 tập tin phụ F1 và F2

- **Giải thuật trộn tự nhiên :**

sodoancon = 0 ;

Lặp lại các bước sau :

- Phân đoạn F thành F1 và F2
- Trộn F1 và F2 vào F

cho đến khi số đoạn con trong F chỉ còn hai

*** GT phân đoạn :**

(1) Mở file F để đọc, mở file F1 và F2 để ghi,

(2) While <chưa hết F> do

(i) Sao 1 đoạn con có thứ tự của F vào F1 như sau:

Nếu chưa hết tập F thực hiện việc đọc PT của F và ghi nó vào F1 cho đến khi gặp PT nhỏ hơn PT trước nó hoặc hết tập F; tăng sodoancon lên 1

(ii) Sao 1 đoạn con có thứ tự của F vào F2 như sau:

Nếu chưa hết tập F thực hiện việc đọc PT của F và ghi nó vào F2 cho đến khi gặp PT nhỏ hơn PT trước nó hoặc hết tập F; tăng sodoancon lên 1

*** GT trộn :**

Trộn các đoạn con có thứ tự trong F1 và F2 vào F

(1) Mở file F1, F2 để đọc, mở file F để ghi

(2) Chừng nào <chưa hết F1> và <chưa hết F2>

(i) Chừng nào <chưa hết đoạn con trong F1> hay <chưa hết đoạn con trong F2> do

- nếu PT x của F1 < PT y của F2 thì sao x vào F và đọc PT tiếp theo của F1 vào x
- nếu PT x của F1 > PT y của F2 thì sao y vào F và đọc PT tiếp theo của F2 vào y

(3) Nếu file F1 chưa kết thúc thì ghi các PT còn lại của nó sang F

ngược lại nếu file F2 chưa kết thúc thì ghi các PT còn lại của F2 sang F

* Chú ý : Trong GT phân đoạn hoặc trộn ta có thể dùng mảng hoặc danh sách liên kết để thay thế cho các file F1 và F2 nếu không gian bộ nhớ trong cho phép. ĐỘ PHỨC TẠP của GT này là $O(n\log_2 n)$

*** Cài đặt :**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include<conio.h>
typedef struct{
 FILE *f;
 int buffer;
}tape ;
void opentape(tape *X, const char *fn)
{
 (*X).f=fopen(fn,"rb");
 fread(&(*X).buffer,sizeof(int),1,(*X).f);
}
void readtape(tape *X,int *x)     //doc du lieu tu
X.buffer vao bien x
{
 memcpy(&*x,&(*X).buffer,sizeof(int));
 fread(&(*X).buffer,sizeof(int),1,(*X).f);
}
void copyitem(tape *X,tape *Y,int *EOR)  //doc ban ghi
{
```

```
 int x;
 readtape(&*X,&x);
 fwrite (&x,sizeof(int),1,(*Y).f);
 *EOR=feof((*X).f) || (x>(*X).buffer);
}
void copyrun(tape *X, tape *Y)  //copy doan con
{
 int EOR;
 do{
  copyitem(&*X,&*Y,&EOR);
 }while(!EOR);
}
void distribute(tape *a,tape *b,tape *c)  //phan doan
{
 r=0;
 do{
  copyrun(&*a,&*b);r++;
  if(!feof((*a).f)) { copyrun(&*a,&*c);r++; }
 }while(!feof((*a).f));
}
void mergerun(tape *a,tape *b,tape *c)  // tron
{
 int EOR;
 do{
  if((*b).buffer<(*c).buffer)
  {
   copyitem(&*b,&*a,&EOR);
   if(EOR)
    copyrun(&*c,&*a);
  }
  else
  {
   copyitem(&*c,&*a,&EOR);
   if(EOR) copyrun(&*b,&*a);
  }
 }while(!EOR);
}
void merge(tape *a, tape *b,tape *c)  //tron
{

 while((!feof((*b).f))&&(!feof((*c).f)))
 {
  mergerun(&*a,&*b,&*c);

 }
    while(!feof((*b).f))
```

```
 {
  copyrun(&*b,&*a);

 }
 while(!feof((*c).f))
 {
  copyrun(&*c,&*a);
 }
}
void natmerge(const char *fn)   //tron tu nhien
{
 long r;
 tape a,b,c;
 do{
  opentape(&a,&*fn);
  b.f=fopen("h1.txt","wb");
  c.f=fopen("h2.txt","wb");
  distribute (&a,&b,&c);
  fclose(a.f);
  fclose(b.f);
  fclose(c.f);
  a.f=fopen(fn,"wb");
  opentape(&b,"h1.txt");
  opentape(&c,"h2.txt");
  merge(&a,&b,&c,&R);
  fclose(a.f);
  fclose(b.f);
  fclose(c.f);
 }while(r!=2);
 remove("h1.txt");
 remove("h2.txt");
}
void main()
{
 FILE *myfile;
 int RANGE_MIN = 0;//hai bien nay dung de gioi han mien so
ngau nhien
    int RANGE_MAX = 1000;
 int rec,ope,sopt=10;//rec-bien dung de ghi vao file
nguon; ope-bien dung de doc, kiem tra file nguon;sopt-
bien gioi han so phan tu tao ra o file nguon
//de thi du toi se tao ra 100 so ngau nhien trong khoang
0->1000
  clrscr();
  randomize();
 if( (myfile = fopen( "c:\mao.txt", "wb" )) != NULL )
```

```
 {
  for(int i=0;i<sopt;i++)
  {
   rec=random(100);
   fwrite(&rec,sizeof(int),1,myfile);
  }
  fcloseall();
 }
// doc ra kiem tra tap tin nguon---->
 if( (myfile = fopen( "c:\mao.txt", "rb" )) != NULL )
 {
  for(int j=0;j<sopt;j++)
  {
   fread(&ope,sizeof(int),1,myfile);
   cout<<ope<<"  ";
  }
  fcloseall();
 }
//<----doc ra kiem tra tap tin nguon
 cout<<"\n";
 natmerge("c:\mao.txt");//bat dau tron

//xuat ket qua
 if( (myfile = fopen( "c:\mao.txt", "rb" )) != NULL )
 {
  for(int k=0;k<sopt;k++)
  {
   fread(&ope,sizeof(int),1,myfile);
   cout<<ope<<"  ";
  }
  fcloseall();
 }
}
```

B. Tìm kiếm trên tập tin :

IV. Khái niệm : File truy nhập trực tiếp là file có thể truy nhập một cách trực tiếp vào mỗi thành phần bằng cách đặc tả vị trí của nó trong file. Ta có thể tưởng tượng file này như 1 mảng rất lớn được lưu trữ trong bộ nhớ ngoài thay vì bộ nhớ trong.

V. Kỹ thuật tìm kiếm :

V.1 Tìm kiếm tuyến tính :

Tìm kiếm tuần tự từ bản ghi đầu tiên đến bản ghi cuối hoặc đến vị trí tìm thấy giá trị muốn tìm.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
```

```
#include<conio.h>

FILE *myfile; int sopt;

//Tao file
void taofile()
{
 cout<< '\n' <<"Nhap so phan tu cua file:" ;
 cin>>sopt;
 int rec;
  randomize();
 if( (myfile = fopen( "ph.txt", "wb" )) != NULL )
 {
  for(int i=0;i<sopt;i++)
  {
   rec=random(100);
   fwrite(&rec,sizeof(int),1,myfile);
  }
  fcloseall();
 }
}
//Doc file
void docfile()
{
  int x;
  if( (myfile = fopen( "ph.txt", "rb" )) != NULL )
 {
  for(int k=0;k<sopt;k++)
  {
   fread(&x,sizeof(int),1,myfile);
   cout<<x<<"  ";
  }
  fcloseall(); getch();
 }
 }
//Tim kiem
void timtuyentinh()
{
  int x,s,found=1;
 cout << '\n' <<"nhap so muon tim:";
 cin >>x ;
 if( (myfile = fopen( "ph.txt", "rb" )) != NULL )
 {
  for(int k=0;k<sopt;k++)
  {
   fread(&s,sizeof(int),1,myfile);
```

```
   if (s==x) {cout<<"So " <<s<<" co o vi tri "<<k+1
<<'\n';found=0;break;}
  }
  if (found) cout<<'\n'<<"Khong tim thay "<<x<<" trong
file" ;
  fcloseall();getch();
 }
}

void main()
{
 clrscr();
 taofile();
 docfile();
 timtuyentinh();
}
```

BÀI TẬP Cấu trúc dữ liệu và Giải thuật

Chương 1 Tổng quan về Cấu trúc dữ liệu và Giải thuật

Bài 1 : Tính biểu thức $s = x^n$ với x, n nhập từ bàn phím. Xác định độ phức tạp của thuật toán.

Bài 2 : Tính :

a) $s = 1 + 1/2 + 1/3 + ... + 1/n$

b) $s = 1 + 1/2! + 1/3! + ... + 1/n!$

c) $s = 1 = 1/3! + 1/5! + ..$

d) $S = x - x^3/3! + x^5/5! - ... + (-1)^n x^{2n-1} / (2n-1)! = Sin(x)$

e) $S = x + x^3/3! + x^5/5! - ... + x^{2n+1} / (2n-1)! = Sh(x)$

f) $S = 1 - x^2/2! + x^4/4! - ... + (-1)^n . x^{2n} / (2n)! = Cos(x)$

g) $S = 1 + x^2/2! + x^4/4! + ... + x^{2n} / (2n)! = Ch(x)$

Bài 3 : Tính n!! (n!! = 1.3.5...n nếu n lẻ và n!! = 2.4.6...n nếu n chẵn)

Bài 4 :

$$S = \sum_{i=1}^{n} (i!!)^{(-1)^i} \qquad \text{với } n > 1$$

Bài 5 :

Một đường thẳng có thể được biểu diễn bởi phương trình :

$ax + by + c = 0$ (a, b, c là các số thực)

Viết chương trình đọc hai bộ ba số thực (tức 6 số thực) trên hai hàng tương ứng với hai phương trình của đường thẳng. CT hiển thị thông báo về mối quan hệ giữa hai đường thẳng ấy. Các thông báo có thể là :

'Hai đường thẳng cắt nhau',

'Hai đường thẳng cắt nhau và vuông góc nhau',

'Hai đường thẳng song song với nhau',

'Hai đường thẳng trùng nhau'.

Bài 6 :

Viết chương trình hiển thị :

- Các số hoàn thiện nhỏ hơn 1000.
- Các số nguyên tố trong khoảng [a, b] với a, b nhập từ bàn fím.

Bài 7 :

Viết chương trình (CT) để tính diện tích (DT) các hình tròn, hình chữ nhật, và hình tam giác theo cách sau đây :

- Nếu nhập 1 số dương , CT hiển thị DT hình tròn có bán kính là số vừa nhập.
- Nếu nhập 2 số dương trên cùng 1 hàng, CT hiển thị DT hình chữ nhật có chiều dài và rộng là 2 số vừa nhập.

- Nếu nhập 3 số dương trên cùng 1 hàng, CT hiển thị DT hình tam giác có chiều dai 3 cạnh là 3 số vừa nhập.

Dùng 3 hàm để tính diện tích các hình trên.

Bài 8 :

Viết chương trình đọc 1 chuỗi trên một dòng rồi sau đó hiển thị mỗi từ trên 1 hàng (theo thứ tự xuất hiện), và số các từ trong câu ấy.

Bài 9 :

- Viết chương trình để kiểm tra một chuỗi có phải là palindrome không ?

Tư "MADAM" là 1 palindrome.

- Viết chương trình để kiểm tra hai từ được nhập có phải là anagram của nhau không ? Ví dụ các từ dear, read là anagram của nhau.

Bài 10 :

Viết chương trình hiển thị các số từ 50 đến 100 có biểu diễn nhị phân là đối xứng. Ví dụ các số 1, 3, 5, 7, 9 là các số có biểu diễn nhị phân là đối xứng, vì biểu diễn nhị phân của chúng lần lượt là 1, 11, 101, 111, 1001.

Bài 11 :

Cho các ma trận vuông cấp n : A, B, C. Viết chương trình con thực hiện các công việc sau :

a) Hoán vị dòng thứ i và dòng thứ k của ma trận nếu i <= n và k <= n.

b) Tính giá trị x1, x2, ..., xn nếu xi bằng tổng các phần tử cột thứ i của ma trận A.

c) Tính các giá trị của dik (i, k = 1,2,...,n) theo công thức dik = max (Aik, Bik, Cik)

Bài 12:

Dùng giải thuật đệ qui để liệt kê :

a) tất cả các hoán vị của n số tự nhiên đầu tiên

b) tất cả các chuỗi nhị phân có độ dài n.

Bài 13 :

Viết chương trình sắp xếp dãy sỏi 3 màu theo thứ tự Đỏ, Xanh, Vàng (mỗi lần chỉ được đổi chỗ 2 viên sỏi). Nhập dãy sỏi ban đầu, đổi chỗ và in kết quả

Chương 2 Cấu trúc mảng

Bài 1 :

Viết chương trình con để thực hiện các phép toán cơ bản về mảng các số nguyên như sau :

a) Tạo mảng bằng cách :

*) nhập các số từ bàn phím

*) lấy ngẫu nhiên các phần tử không trùng nhau

*) lấy ngẫu nhiên các số mà từng cặp số liền nhau không trùng nhau

b) In mảng ra màn hình theo trật tự :

*) bình thường

*) số chẵn, số lẻ, các số bằng 0

*) các số 0, số âm, số dương

c) Chèn vào mảng 1 số x nào đó :

*) ở vị trí thứ n cho trước

*) ở trước hoặc sau 1 số y nào đó

*) nếu mảng đã sắp xếp, thì sau khi chèn x vào, mảng vẫn sắp xếp

*) chèn x vào tại nhiều vị trí thỏa mãn điều kiện nào đó

d) Xóa số trong mảng

*) bằng 1 số x nào đó (nếu có)

*) thỏa mãn 1 điiều kiện nào đó

e) Tìm kiếm trên mảng :

*) tìm tuyến tính

*) tìm nhị phân : điều kiện, thuật giải

f) Sắp xếp mảng :

*) bằng phương pháp chọn, chèn, nổi bọt : giải thuật, minh họa với dãy số cho trước, cài đặt bằng Pascal.

*) bằng phương pháp phân hoạch (QuickSort) : đệ qui và không đệ qui

g) Thay thế 1 phần tử của danh sách bởi 1 phần tử khác

Bài 2 :

Sử dụng đệ qui và không đệ qui để viết hàm kiểm tra 1 mảng có đối xứng hay không ?

Bài 3 :

a) Hãy trộn 2 mảng thành 1 mảng mới theo nguyên tắc từng đôi một.

b) Trộn nhiều mảng thành 1 mảng mới

c) Trộn 2 mảng đã được sắp xếp thành 1 mảng cũng được sắp xếp

d) Tách 1 mảng thành nhiều mảng theo 1 nguyên tắc nào đó

Bài 4 :

Viết giải thuật sắp xếp mảng và đồng thời tỉa bớt những phần tử giống nhau của mảng, chỉ giữ lại một phần tử làm đại diện.

Bài 5 :

Tìm các số nguyên tố trong mảng.

Chương 3 Danh sách liên kết

Bài 1 :

1) Tạo một danh sách liên kết các số nguyên bằng cách lấy ngẫu nhiên

2) Đếm các số có trong danh sách và tìm giá trị trung bình của các phần tử trong DS

3) Viết hàm để xác định tính tăng dần của DS

4) Viết chương trình con nối 1 nút vào DS

5) Viết CT con để xóa 1 phần tử thứ n trong DS sau đó gọi Ct con này để xóa tất cả các phần tử chia hết cho 4

6) Viết CT con để chèn 1 phần tử cho trước vào sau nút thứ n trong DS sau đó gọi Ct con này để chèn thêm các phần tử vào sau tất cả các phần tử chẵn của DS

Bài 2 :

1) Viết CT nối 2 DSLK, chỉ thay đổi liên kết và chấp nhận phá vỡ 2 DS ban đầu

2) Viết CT trộn 2 DSLK có thứ tự tăng dần và tạo DSLK thứ 3 cũng tăng dần, trong đó 2 DS ban đầu được bảo toàn

Bài 3 : Một đa thức có thể được biểu diễn như 1 DSLK với mỗi nút sẽ chứa hệ số và số mũ của từng thành phần của đa thức.

1) Nhập đa thức vào DS(có tham số hình thức)
2) Cộng, trừ hai đa thức
3) In kết quả

Bài 4 : Nhập 2 danh sách liên kết (DSLK), sau đó in ra màn hình :

1) Phần giao của 2 DSLK
2) Phần hội của 2 DSLK
3) Hiệu của DSLK thứ nhất và DSLK thứ hai

Bài 5 :

Viết chương trình con để đảo ngược một danh sách liên kết trong 2 trường hợp :

a) chỉ đảo ngược dữ liệu
b) thay đổi mối liên kết

Bài 6 :

Người ta muốn cho đăng ký để bán vé cho 1 buổi hòa nhạc, ai đăng ký trước sẽ được mua trước. Hãy viết 1 chương trình đọc các tên và địa chỉ của những người đăng ký vé cùng số vé họ yêu cầu và lưu trữ chúng trong DSLK. Chương trình tạo ra một danh sách các tên, địa chỉ, và số vé của những người được mua. Lưu ý là không có người nào được đăng ký nhiều lần.Bài **Bài 7 :**

Hãy viết hàm trả lại một con trỏ chỉ đến nút cuối cùng trong 1 DSLK.(đệ qui và không đệ qui)

Bài 8 :

Hãy viết hàm đếm số nút trong 1 DSLK (đệ qui và không đệ qui)

Bài 9 :

Giả sử ta đã có File HS.dat chứa dữ liệu là các bản ghi, mà mỗi bản ghi gồm tên, số hiệu lớp và điểm trung bình của từng học sinh. Hãy lập nhiều DSLK chứa các bản ghi ấy, một DS cho 1 lớp. Mỗi DS phải được sắp xếp theo vần abc của tên. Sau khi lập xong hãy in ra với những đầu đề thích hợp.

Chương 4 Ngăn xếp và Hàng đợi

Bài 1 :

Viết 1 chương trình dùng các phép toán trên ngăn xếp như Empty, Creat, Push, Pop để :

a) Tìm phần tử ở đỉnh ngăn xếp, nhưng không xóa nó từ ngăn xếp
b) Tìm phần tử ở đáy ngăn xếp, và để lại ngăn xếp rỗng
c) Tìm phần tử thứ n của ngăn xếp, để lại NX không có n PT ở trên
d) Tìm phần tử thứ n của ngăn xếp, nhưng vẫn bảo toàn NX
e) Tìm phần tử ở đáy ngăn xếp, và bảo toàn NX

Bài 2 :

Viết chương trình để xác định 1 biểu thức (là 1 chuỗi kí tự) có chứa các dấu ngoặc tương xứng không, nghĩa là mỗi dấu ngoặc trái phải có 1 dấu ngoặc phải tương ứng.

Bài 3 :
Viết chương trình để xác định các thừa số nguyên tố của 1 số nguyên dương lớn hơn 2, nhưng in ra theo thứ tự giảm dần. Ví dụ 18 = 3 * 3 * 2

Bài 4 :
a) Viết chương trình tính giá trị biểu thức hậu tố, giả sử các toán hạng chỉ có 1 chữ số không âm

Ví dụ : đầu vào : 1 2 3 + * 1 2 - / { / : div }
1 3 $ 5 + * 1 2 - / { kt trống và $: bỏ qua }
1 3 + + { báo thừa phép toán }
2 3 4 5 + { báo thừa toán hạng }

b) Viết chương trình tính giá trị biểu thức hậu tố, giả sử các toán hạng có thể có nhiều hơn 1 chữ số. Giả sử các toán hạng và các toán tử cách nhau 1 kí tự trống.

Ví dụ 12 2 + 15 192 - * { (12 + 2) * (15 - 192) }

Bài 5 :
Thiết kế giải thuật và viết chương trình chuyển biểu thức trung tố sang biểu thức hậu tố. Có thể dùng hàng đợi để lưu trữ biểu thức hậu tố.

Bài 6 :
Viết hàm (hoặc thủ tục) xác định 1 biểu thức hậu tố được viết đúng hay không ?

Bài 7 :
Viết 1 thủ tục dùng các phép toán trên hàng đợi (HĐ) như EmptyQ, CreatQ, AddQ, RemoveQ để :
a) Tìm phần tử ở đầu HĐ, nhưng không xóa nó từ HĐ
b) Tìm phần tử ở cuối HĐ, và để lại HĐ rỗng
c) Tìm phần tử thứ n của HĐ, để lại HĐ không có n PT đầu tiên
d) Tìm phần tử thứ n của HĐ, nhưng vẫn bảo toàn HĐ

Bài 8 :
Dùng các phép toán cơ bản của HĐ và NX, viết thuật toán và cài đặt CT đảo ngược các PT của HĐ.

Bài 9 :
Viết CT đọc 1 chuỗi kí tự, đẩy mỗi kí tự vào NX theo thứ tự như khi chúng được đọc và đồng thời thêm nó vào HĐ. Khi đến kết thúc chuỗi, dùng các phép toán cơ bản của NX và HĐ để xác định chuỗi đó có phải là 1 palindrome không ?(Vd chuỗi "madam" là 1 palindrome)

Bài 10 : Sắp xếp mảng bằng phương pháp RadixSort (dùng HĐ)

Chương 5

CÂY

Bài 1 : Với mỗi danh sách từ khóa Pascal dưới đây :
a) Vẽ cây tìm kiếm nhị phân tạo ra khi các từ được chèn theo thứ tự đã cho
b) Thực hiện kiểu quét trung tự, hậu tự và tiền tự cho cây
(i) program, const, type, function, procedure, begin, end
(ii) div, mod, array, for, to, do, repeat, until, with, while, label, goto

Bài 2 : Bắt đầu với cây nhị phân sau, hãy chỉ ra cây BST nhận được sau khi thực hiện dãy các thao tác sau :
a) Chèn 7, 1, 55, 25
b) Xóa 8, 37, 62
c) Chèn 7, xóa 8, chèn 59, xóa 60

Bài 3 : Đối với cây ở bài tập 2, hãy hiển thị kết quả tạo bởi việc quét cây theo kiểu trung tự, tiền tự và hậu tự.

Bài 4 :
Với mỗi biểu thức số học dưới đây, hãy vẽ cây nhị phân biểu diễn biểu thức ấy rồi dùng các kiểu quét để tìm biểu thức trung tố và hậu tố tương đương :
a) (A - B) - C
b) A - (B - C)
c) A / (B - (C - (D - (E - F))))
d) ((((A - B) - C) - D) - E) / F
e) ((A * (B + C)) / (D - (E + F))) * (G / (H / (I * J)))

Bài 5 :
Dùng cài đặt trên cơ sở mảng của cây tìm kiếm nhị phân mô tả ở phần này, hãy chỉ ra nội dung của mảng lưu trữ BST ở bài tập 3.

Bài 6 : Hãy vẽ cây nhị phân với điều kiện sau :
a) Kiểu quét trung tự của cây nhị phân tạo ra : G F H K D L A W R Q P Z
và kiểu quét tiền tự tạo ra : A D F G H K L P Q R W Z
b) Kiểu quét hậu tự của cây nhị phân tạo ra : F G H D A L P Q R Z W K
và kiểu quét trung tự tạo ra như ở a)

Bài 7 :
a) Viết hàm đệ qui để tìm số nút lá trong 1 cây nhị phân
b) Viết hàm đệ qui để tìm số nút bậc hai trong 1 cây nhị phân
c) Viết hàm xác định mức của cây
d) Viết hàm xác định mức của một nút cho trước

Bài 8 : Viết thủ tục **không** đệ qui để quét cây kiểu trung tự (dùng 1 NX chứa các con trỏ để loại trừ phép đệ qui)

Bài 9 :
Viết thủ tục để quét cây theo từng mức (thủ tục không đệ qui, và dùng 1 HĐ các con trỏ)

Bài 10 : Viết chương trình xử lý BST có các nút chứa kí tự. Người dùng được phép chọn lựa theo menu dưới đây :

I : chèn 1 kí tự vào cây
S : tìm 1 kí tự cho trước
TI : quét kiểu trung tự
TP : quét kiểu tiền tự
TR : quét kiểu hậu tự
D : xóa 1 kí tự
Q : kết thúc

Bài 11 :

Viết thủ tục sắp xếp theo kiểu cây của 1 danh sách các số nguyên được lưu trữ trong :

a) một mảng
b) một danh sách liên kết

Bài 12 :

Viết chương trình **kiểm tra từ**, nghĩa là đọc các từ trong một phần văn bản rồi tìm chúng trong 1 tự điển. Dùng 1 BST để lưu trữ tự điển này, đọc danh sách các từ trong 1 tập tin. Khi kiểm tra các từ trong 1 phần văn bản , chương trình in ra danh sách tất cả các từ không có trong tự điển.

Bài 13 :

Trong 1 công ty, phương pháp trả lương cho mỗi nhân viên (NV) tùy thuộc vào NV đó là NV hành chính, công nhân hay NV bán hàng. Giả sử rằng 1 tập tin các bản ghi NV được bảo trì trong đó mỗi bản ghi chứa các thông tin sau cho mỗi NV : Tên (20 kí tự) , số bảo hiểm xã hội (số nguyên), tuổi (số nguyên), số người còn phụ thuộc (số nguyên), mã NV (các kí tự O, F, S biểu diễn NV hành chính, công nhân và NV bán hàng theo thứ tự đó), lương mỗi giờ nếu là công nhân, lương hằng năm nếu là NV hành chính, lương cơ bản và số phần trăm hoa hồng nếu là NV bán hàng.

Viết chương trình điều khiển bằng menu cho phép ít nhất các tùy chọn sau đây đối với người dùng :

GET : đọc các bản ghi từ tập tin nhân viên và lưu trữ chúng trong 1 BST, được sắp xếp theo tên.

INS : chèn 1 bản ghi cho 1 NV mới vào BST

UPD : cập nhật bản ghi của 1 NV có trong cây

RET : tìm và hiển thị bản ghi của 1 NV cho trước (bằng tên hay bằng số bảo hiểm xã hội)

LIS : In các bản ghi theo thứ tự. Tùy chọn này cho phép các tùy chọn con sau:

ALL : In tất cả các NV
OFF : chỉ in các NV hành chính
FAC : chỉ in các công nhân
SAL : chỉ in các NV bán hàng

DEL : xóa 1 bản ghi của 1 NV từ BST

SAV : Sao các bản ghi vào tập tin

QUI : kết thúc

Chương 6 & 7

Kỹ thuật băm
Tìm kiếm và Sắp xếp ngoài

Bài 1 :

Dùng phương pháp dò tuyến tính tạo bảng băm cho 10 số ngẫu nhiên trong khoảng 0..99

Bài 2 :

Tạo một từ điển các từ khóa của NNLT Pascal bằng BST, bằng mảng các xích của bảng băm. Viết chương trình tìm một từ trong từ điển, xóa một từ và thêm một từ vào từ điển. Dữ liệu được lưu trong file.

Bài 3 :

Sử dụng bảng băm để kiểm tra tính hợp lệ của các định danh người dùng và các mật khẩu cho 1 hệ máy tính.Một danh sách các định danh người dùng và các mật khẩu được đọc từ 1 tập tin định kiểu đã có và được lưu trữ trong 1 bảng băm. Khi đưa vào định danh người dùng và mật khẩu, chúng sẽ được tìm trong bảng băm này để xác định có hợp lệ không ?

Bài 4 :

Viết chương trình cài đặt thuật toán sắp xếp tập tin bằng phương pháp trộn tự nhiên

Bài 5 :

a) Viết chương trình trộn 2 tập tin đã có thứ tự thành 1 tập tin cũng có thứ tự

b) Sử dụng câu a) để viết chương trình trộn nhiều tập tin đã có thứ tự thành 1 tập tin cũng có thứ tự.

TÀI LIỆU THAM KHẢO

1. Trần Quốc Chiến (1998), Giáo trình *Cấu trúc dữ liệu và giải thuật,* lưu hành nội bộ.
2. Trần Đức Huyên (1997), *Phương pháp giải các các bài toán tin học*, Nxb Giáo dục, Hà Nội.
3. Larry Nyhoff, Sanfort Leedstma (1998), *Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu*, Nxb Đà Nẵng.
4. Đỗ Xuân Lôi (1998), *Cấu trúc dữ liệu và giải thuật*, Nxb Khoa học và kỹ thuật, Hà Nội.
5. Robert Sedgewick (1998), *Cẩm nang thuật toán,* Nxb Khoa học và kỹ thuật, Hà Nội.
6. Nguyễn văn Thân, Phan văn Thảo (1996), *Thuật ngữ Tin học Anh Việt*, Nxb tp Hồ Chí Minh.
7. Vũ Đức Thi (1999), *Thuật toán trong tin học*, Nxb Khoa học và kỹ thuật, Hà Nội.
8. Nguyễn Trung Trực (1996), *Cấu trúc dữ liệu*, Nxb Trung tâm điện toán trường Đại học bách khoa tp Hồ Chí Minh.
9. Lê Minh Trung (1996), *Bài tập Cấu trúc dữ liệu và thuật toán*, Nxb tp Hồ Chí Minh.
10. Nguyễn Thanh Thủy, Nguyễn Quang Huy (1999), *Bài tập lập trình ngôn ngữ C*, Nxb Khoa học và kỹ thuật, Hà Nội.
11. Gerald Leblanc (1995), *Turbo C*, Nxb Khoa học và kỹ thuật, Hà Nội.
12. Huỳnh Tấn Dũng, Hoàng Đức Hải (2004), Bài tập ngôn ngữ C, Nxb Lao động xã hội, tp Hồ Chí Minh.
13. Đinh Mạnh Tường (2003), *Cấu trúc dữ liệu & Thuật toán*, Nxb Khoa học và Kỹ thuật, Hà Nội.
14. Ngô Trung Việt (1995), *Ngôn ngữ lập trình C và C*$^{++}$, Nxb Giao thông vận tải, Hà Nội.

MỤC LỤC